2847

A Gentle Introduction to the VAX™ System

J.R. Hubbard

A GENTLE INTRODUCTION TO THE VAX™ SYSTEM

To A.H.H.,
my favorite user

A GENTLE INTRODUCTION TO THE VAX™ SYSTEM

J. R. HUBBARD

Professor of Mathematics & Computer Science
University of Richmond
Richmond, Virginia

TAB Professional and Reference Books

Division of TAB BOOKS Inc.
P.O. Box 40, Blue Ridge Summit, PA 17214

Gesellschaft für Mathematik und
Datenverarbeitung mbH Bonn

Trademarks
DCL, DEC, DECnet, DECwriter, Digital, LA,
Rainbow, VAX, VMS, and VT are trademarks of the Digital Equipment Corporation.
Ada is a trademark of the U.S. Department of Defense
CP/M is a trademark of Digital Research, Inc.
IBM and IBM PC are trademarks of the International Business Machines Corporation.
Minitab is a trademark of Minitab Inc.
UNIX is a trademark of AT&T Bell Laboratories.
EUNICE is a trademark of The Wollongong Group.

FIRST EDITION
FIRST PRINTING

Copyright © 1987 by J. R. Hubbard
Printed in the United States of America

Reproduction or publication of the content in any manner, without express permission of the publisher, is prohibited. No liability is assumed with respect to the use of the information herein.

Library of Congress Cataloging in Publication Data

Hubbard, J. R. (John Rast), 1943-
A gentle introduction to the VAX system.

Bibliography: p.
Includes index.
1. VAX computers. I. Title.
QA76.8.V32H83 1987 004.1'4 87-5036
ISBN 0-8306-2847-9 (pbk.)

Questions regarding the content of this book
should be addressed to:

Reader Inquiry Branch
Editorial Department
TAB BOOKS Inc.
P.O. Box 40
Blue Ridge Summit, PA 17214

Contents

Preface ix

Introduction xi

1 Your First Session at the Terminal 1

The Terminal 1
Logging In 4
The DCL Mode 6
DCL Commands 7
Self-Test Exercises 9

2 The EDT Line Editor 13

The File Specification 13
Creating and Referencing Files 15
The Interaction of Modes 18
Self-Test Exercises 27

3 Using the DCL to Manipulate Files 29

The DELETE and PURGE Commands 29
The COPY and APPEND Commands 30
Concatenation
The DIRECTORY/SIZE Command 33
Managing Memory
DCL Wildcards 36
Defining Keypad Keys 37
Abbreviated Commands
Self-Test Exercises 39

4 The EDT Keypad Editor 41

The Keypad Mode 41
The Keypad Map
Character Editor Commands 44
The DELETE Commands—Cutting and Pasting—Reformatting—The FIND Commands
Self-Test Exercises 51

5 Running a Pascal Program 55

The Pascal Programming Language 55
A Sample Pascal Program 56
Writing the Program—Compiling the Program—Linking the Program—Running the Program
An Interactive Program 61
Program Variables—Compile-Time Errors—Logical Errors—Run-Time Errors
Self-Test Exercises 66

6 Introduction to Minitab 69

The Minitab Mode 69
The Purpose of Minitab—The Data
The READ Command 71
Column Labels—Examining Your Workspace
Simple Statistics 73
The Histogram—Plotting Data—The Correlation Coefficient—Creating New Columns
The SAVE Command 78
ASCII and Non-ASCII Files—Default Files
Retrieving Data 82
Array Processing—Simulation
Self-Test Exercises 84

7 Running a BASIC Program 87

The Versions of BASIC 87
Compiled and Interpreted BASIC—The VAX BASIC Mode
The BASIC Program 89
Running the Program—Debugging—Editing
The SAVE Command 93
Text Files
Using the EDT Editor 94
Syntax Errors—Logical Errors—Stopping a Program
Self-Test Exercises 99

8 VAX/VMS Subdirectories and Protection Codes 101

Creating Subdirectories 101
The CREATE/DIRECTORY Command—The Default Directory—The Parent Directory
The Tree Diagram 103
Scope—The Asterisk—The Ellipsis
File Protection 108
The User Identification Code—User Categories—Types of Access
Syntax for the SET PROTECTION Command 112
Self-Test Exercises 113

9 Command Procedures 115

Uses for Command Procedures 116
Comments—The SET VERIFY Command—User Interaction—Aborting a Command Procedure—Responding to Errors
The ON . . . THEN Command 120
The Parameter
The Login Procedure 121
String Assignment Statements
Performing Repetitive Tasks 123

Producing Output on the Screen—Interactive Input
Self-Test Exercises 125

10 Transition to UNIX 127

The UNIX Operating System 128
The Native Mode UNIX—The Emulator UNIX
Logging In to UNIX 129
The UNIX Prompt—Fundamental UNIX Commands
File Processing with UNIX 130
The ed Editor—Options—Arguments—UNIX Wildcards
Concatenating Files 134
The grep Command 136
Self-Test Exercises 137

11 The UNIX Text Editor 139

The Line Editor 140
Commands—Syntax
Creating a File 141
Inserting External Files
Deleting and Inserting Files 144
The Substitute Command—The Move Command—The Search Command
Self-Test Exercises 147

Appendix A VAX Modes 151

Appendix B DCL Commands 157

Appendix C EDT Line Commands 163

Appendix D EDT Keypad Commands 169

Appendix E Minitab Commands 175

Appendix F BASIC Commands 183

Appendix G UNIX Shell Commands 187

Appendix H UNIX Text Editing Commands 193

Appendix I MAIL Commands 197

Appendix J The ASCII Code 201

Glossary 205

Bibliography 219

Index 223

Preface

Even while they teach, men learn.

Seneca (Letters)

This book was written originally for students enrolled in computer science courses at the University of Richmond. Although many of these students were experienced with microcomputers, very few had worked on a large time-sharing system like the VAX.

The purpose of this book is to help the novice become comfortable using any of the Digital Equipment Corporation's VAX computers, from the MicroVAX to the powerful VAX 8000 system. The first nine chapters cover systems which run on Version 4 of DEC's VAX/VMS operating system, while the last two introduce the user to the UNIX operating system.

The book is meant to be used as a tutorial. A reader, even one with no computer experience, should be able to sit down at a VAX terminal and carry out the exercises in each chapter. Although no supervision should be necessary, having an experienced user nearby to answer questions would facilitate the process. Self-test exercises are provided at the end of each chapter for reinforcement. The reader is also encouraged to "experiment" with each system, and to utilize the relevant HELP files. Each chapter takes about 60 minutes to be complete.

The first four chapters should be read in their given order. These cover the basics of using the VMS operating system, the Digital Command Language, and the EDT editor. Chapters 5 through 9 are independent of one another, and may be read in any order. The last two chapters might have been called "UNIX for VMS Users," because they introduce the UNIX operating system and its standard text editor. They provide perhaps the only available introduction to UNIX that clearly shows the similarities between these two popular operating systems.

A set of eleven appendixes is also provided for reference. Appendixes B through I list the most common commands used in the various systems covered by the book.

Because this book was written as a primer, it is necessarily incomplete. Each chapter is meant only to provide an introduction to the system covered. Although the appendixes provide convenient supplements to the information presented, the serious user eventually will have to resort to the technical manuals listed in the Bibliography.

I wish to thank my wife and colleague Anita Hubbard for painstakingly proofreading the entire manuscript. I am also indebted to Gilpin Brown, David Doll, Jane Hall, Barbara Hubbard, John B. Hubbard, Thomas Hubbard, Mark Surles, and Julie Worcester for their help with various stages of this work over the past three years. Finally, I wish to mention my Computer Science 251 students, without whom this book would not have been necessary.

Introduction

'Where shall I begin, please your Majesty?' he asked.
'Begin at the beginning,' the King said, gravely,
'and go on till you come to the end:
then stop.'

Lewis Carroll (Alice's Adventures in Wonderland)

In the "good old days," only people with some experience in electronics, and a great deal of determination, used computers. The activity required punching thousands of holes in "IBM cards," and then submitting the resulting deck to the computing center. Much later (often several days), the resulting output could be retrieved. This laborious task would be repeated until, finally, all of the "bugs" had been found and removed from the program.

Fortunately, those "good" old days are now gone forever. Today, people whose technical skills might not even be sufficient to drive a car are using computers to write text files and even original programs. Powerful, high-speed microcomputers can be bought for less than a week's wages. Modems use telephone lines to connect these micros to larger computers, and to vast data banks and information services. Scientists in research laboratories and universities have easy access to computers which analyze their data almost instantly.

Like most mainframe computer systems, the Digital Equipment Corpora-

tion's VAX computers are usually accessed through remote terminals. These television-like boxes are connected to the computer itself, which might be in another building, or even another city. Because the communication between the terminal and the computer is so fast, it is easy for the user to think that the terminal is actually the computer. This misconception is especially common among those who have used stand-alone microcomputers, like an Apple or an IBM-PC.

The VAX, however, is actually a multi-user, time-sharing system. This means that many people might be using the same computer at the same time. Each user has a terminal with its own connection to the VAX. If the terminals are in different rooms, the users might even be unaware that anyone else is using the same computer. This magic is achieved through the computer's *operating system*, which continually, but invisibly, switches users on and off from the computer's *central processing unit* (its "brain").

To use the VAX, you must first have an *account* set up by the authority which operates the VAX. Once your account has been created, you will be given a *username* and a *password*. Your username is a unique combination of letters which identifies you to the VAX/VMS system. It will be the first thing you type each time you "log in" to the system. Your password exists for your protection, to prevent unauthorized people from using your account. It is the second thing you type whenever you log in.

The VAX computers at the University of Richmond are controlled by the Academic Computing Center. Each student enrolled in a course which uses a VAX is given an account at the beginning of the semester. Students' usernames are simply their last names followed by as many letters of their first names as it takes to make a total of nine letters. For example, Ronald Reagan would be given the username "REAGANRON." The student's password is identical to the username.

In addition to getting your password and username, you should also find out where the terminals to your VAX are located and what hours you may use them. Use the following spaces to record this essential information for your own reference.

Username:
Password:
Location of Terminals:
Hours:

Now you have all the information you need to begin using the VAX. If you have the opportunity to work with an experienced user, plan to do so. If not, try to arrange to work with another novice. Two heads are better than one. But, even on your own, you will find that the VAX is a very "user-friendly" computer. You should be able to work through this entire tutorial on your own without any trouble. Plan to spend about 60 minutes on each chapter.

Chapter 1

Your First Session at the Terminal

A journey of a thousand miles begins with a single step.

Ancient Proverb

In this chapter, you will learn how to use the video display terminal to access the VAX and its VMS operating system. Work through each step, one at a time. If you have problems along the way, ask someone nearby, or try repeating the last few steps or instructions. If you really get stuck, you can always start over from the beginning. In any case, remember: nothing you type can hurt the computer.

THE TERMINAL

You should now be sitting in front of a video display terminal. The terminal consists of a box with a television-like screen (sometimes called a "CRT," for "Cathode Ray Tube"), connected to a typewriter-like keyboard.

Step 1. Look at the screen for the cursor. It is either a little rectangular light or an underline, blinking on the screen. If you don't see the cursor, the terminal is probably turned off. In this case, flick the toggle switch at the back of the CRT to turn it on. When it is turned on, the terminal will give a little "beep." Then, after it warms up, the blinking cursor will appear on the screen at the upper left-hand corner. If you still don't see the cursor, check to see if the cables in the back of the terminal are plugged in all the way. If that does not solve the problem, try another terminal.

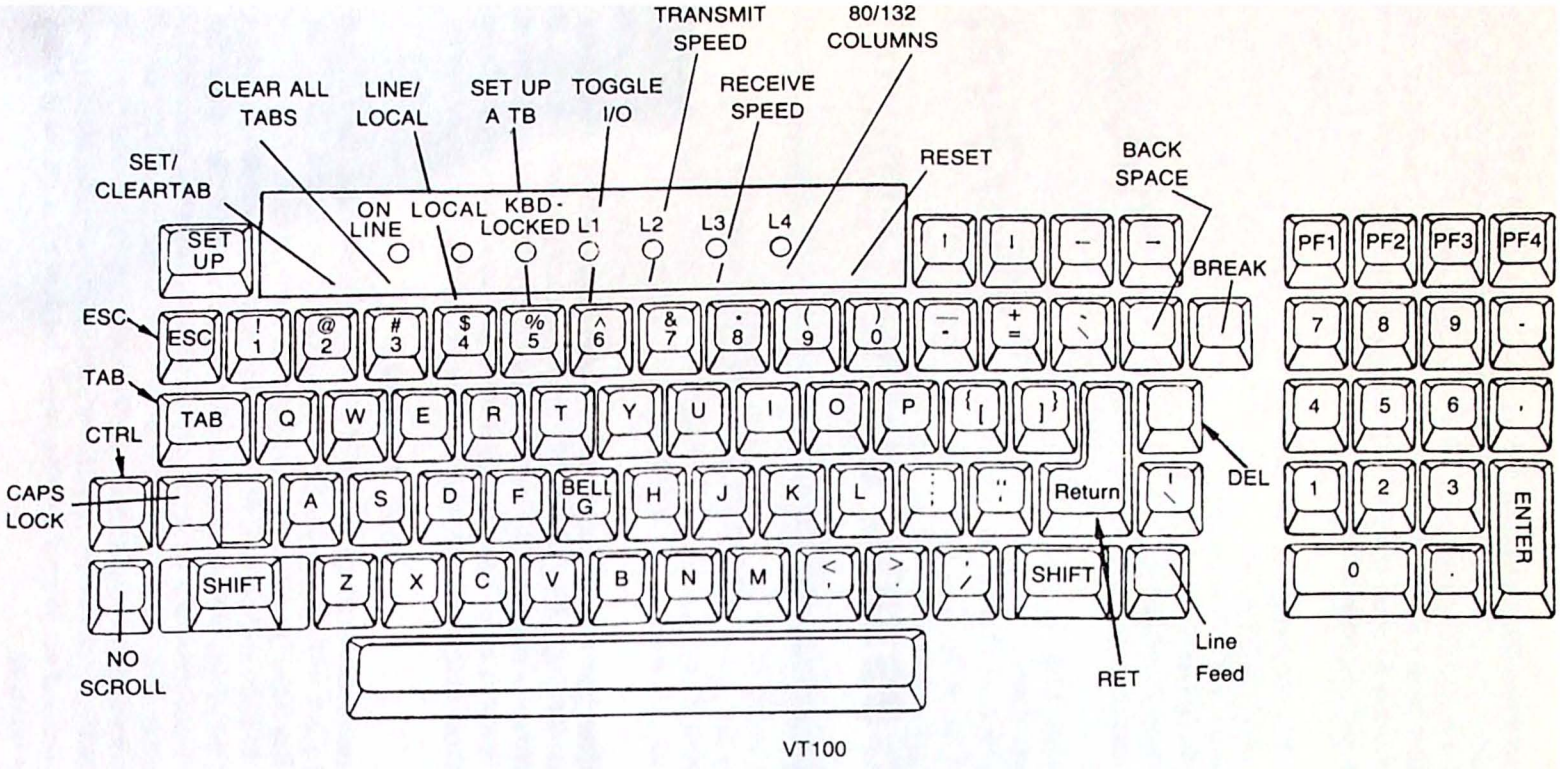

Fig. 1-1. The VT100 Keyboard.

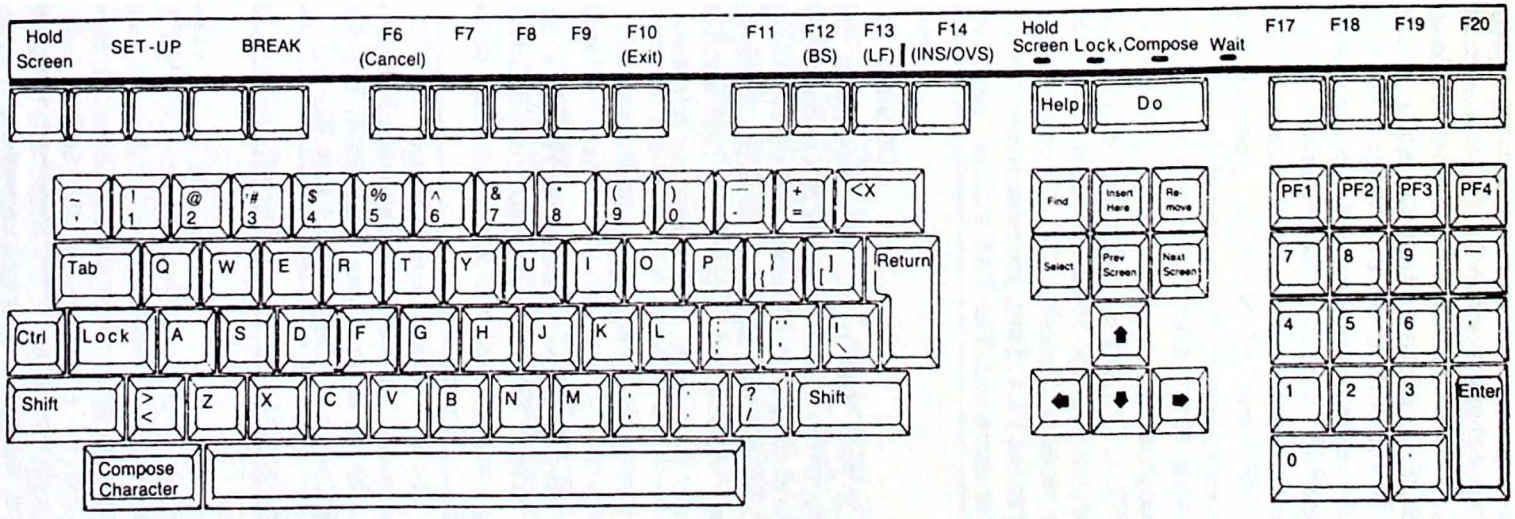

Fig. 1-2. The VT200 Keyboard.

Step 2. Now look at the keyboard. If your terminal is a DEC VT100, the keyboard should look like Fig. 1-1. If your terminal is one of DEC's VT2xx series (e.g., VT220), then the keyboard should look like Fig. 1-2. Any other kind of terminal should have a keyboard which is very similar to these two standards.

A computer terminal keyboard is similar to a typewriter keyboard. In fact, the letter keys are arranged the same way as those on a typewriter (the "QWERTY" format), to facilitate touch-typing. The main difference is the presence of many extra keys. Some of these keys are labeled with words or phrases like "PF1" and "NO SCROLL." Four keys have arrows on them. The labels for the twenty keys on the top row of the VT220 are on a strip just above the keys.

One of the most important keys on the keyboard is the RETURN key. Throughout this book, the RETURN key will be designated by the symbol <RET>. This key is used to send information from your terminal to the VAX. Whatever you type at the keyboard is stored in a small "buffer" memory inside your terminal. Pressing <RET> causes the contents of this local buffer to be sent to the VAX. Thus, you must remember to press <RET> at the end of each line or command. The VAX will not "hear" you until you do.

LOGGING IN

If your VAX is part of a network of computers, or if your terminal is connected to a VAX by means of a telephone system, you probably will have to perform some kind of connect procedure before you can log in. At the University of Richmond, the user must press the BREAK key and then respond by typing either "vax750" or "vax785." If your installation does require some kind of connect procedure, write it here for reference.

Connect Procedure:

Step 3. If your system does require a connect procedure, do that now.

Step 4. Whether your system requires a connect procedure or not, the next thing you need to do is to get the attention of the VAX. This is done by pressing <RET>. Do this now.

The VAX should respond to your <RET> signal by displaying the prompt "Username:" on the screen, although it might display some other information first (like the name of your installation). If you are connected and have pressed <RET>, your screen should look like Fig. 1-3.

```
                    Academic Computing
                University of Richmond
                        VAX 750

Username:
```

Fig. 1-3. The screen, ready for a username.

The cursor should be blinking right after the colon. The VAX has "prompted" you for your username. You should respond by typing your username (not the President's) and then pressing <RET>.

The VAX will then ask you for your password. Type it and press <RET> again. The characters which you type here are not "echoed" on the screen (Fig. 1-4). This is to protect your account in case someone happens to be looking over your shoulder. Notice that the characters shown in color in Fig. 1-4 are those which you should type. Also remember that <RET> must be pressed at the end of each line; this is not indicated on the screen.

```
                                Academic Computing
                            University of Richmond
                                    VAX 750

Username: REAGANRON
Password:
```

Fig. 1-4. The screen, ready for a password.

If you make a typing mistake, use the DELETE key to erase the error. This key, which will be denoted throughout this book by <DEL>, is labeled "DELETE" on the VT100 keyboard (next to the RETURN key). On the VT220 keyboard, it is labeled with an "X" inside a pentagon, and is directly above the RETURN key. Do not use the BACK SPACE key or any of the keys labeled with arrows at this time.

Step 5. At this point, you are in one of several possible situations. If everything has worked out right, there should be a dollar sign ($) displayed on your screen, with the cursor blinking happily just to the right of it. If so, you may proceed directly to Step 6. If you didn't respond fast enough to the "Username:" prompt or to the "Password:" prompt, then the VAX responded with the message shown in Fig. 1-5.

```
Error reading command input
Timeout period expired
```

Fig. 1-5. The screen after taking too much time. The VAX gets impatient after about 18 seconds.

During its log-in procedure, the VAX will give you about 18 seconds to respond to its prompts. If you exceed this limit and get the response shown in Fig. 1-5, just start over again. If you were disconnected from your system, go back to Step 3. Otherwise, go back to Step 4. The only time the VAX requires an immediate response is during its log-in procedure. After you have logged in successfully, you may take as long as you want to respond to prompts.

Another possible problem is that the VAX would not accept your username and password. If so, it would display the message "User authorization failure," as in Fig. 1-6. In this case, just start over with Step 3 or 4.

```
                                Academic Computing
                            University of Richmond
                                    VAX 750

Username: REAGANRON
Password:
User authorization failure
```

Fig. 1-6. The screen after typing the wrong password. Don't forget your password.

If nothing happens when you try to type, your screen is probably "frozen." This is caused by touching the NO SCROLL key on the VT100 keyboard (bottom left corner) or the HOLD SCREEN key on the VT220 keyboard (upper left corner). Just press the key again to "unfreeze" the screen. These keys are used to cause the screen to pause when the information being scrolled across it is coming too fast.

If anything else happened, the best thing to do would be to go back to Step 1, or to ask someone nearby for help.

Step 6. If everything has gone right so far, the system responded to your password with a report about your "last interactive login," possibly some other messages from the computing center, and finally a dollar sign and the blinking cursor. If the dollar sign has not appeared, something has gone wrong. You should either start over with Step 3 or ask someone for help. When the dollar sign appears, you know you have logged in successfully. Congratulations; you are now a VAX "user."

THE DCL MODE

The dollar sign is the VAX's **DCL prompt.** A "prompt" is a symbol which the system displays to let you know that it is waiting for you to type something. For example, the prompt "Username:" was to tell you to type in your username. The prompt "$" is to tell you that you are in the VAX's DCL Mode, in which the system is able to carry out commands which are defined by the VAX's Digital Command Language. These are called DCL commands.

One of the most important DCL commands is the command that allows you to change your password. Every user account has a password, which must be given before you can log in to that account. To keep your password confidential, you should change it regularly. This requires the SET PASSWORD command.

Step 7. Think of a new password. Create some combination of letters and numerals which will be easy for you to remember, but also unlikely for anyone else to guess; for example, your mother's maiden name, or your Social Security Number backwards. Be imaginative!

Suppose you decide to use 388BG2L11111W as your new password. Also, suppose that your current password is REAGANRON (the same as your username). You would change your password as indicated in Fig. 1-7.

```
$ SET PASSWORD
Old password: REAGANRON
New password: 388BG2L11111W
VERIFICATION: 388BG2L11111W
```

Fig. 1-7. The screen after changing a password. The actual passwords you type will not appear on the screen.

First, type SET PASSWORD and press <RET>. Then, type your current password in response to the "Old password:" prompt, and press <RET>. Enter the new password two times, first in response to the "New password:" prompt, and then in response to the "Verification:" prompt. (In this book, the verb "enter" will always mean to type the information and then press the <RET> key.) Notice that the VAX does not echo your passwords. After you have entered your new password in response to the "Verification:" prompt, you will need it every time you log in, so don't forget it.

Although the VAX/VMS operating system does not distinguish between uppercase (capital) and lowercase letters, the DCL commands will always be presented in uppercase. If you want everything you type to be in uppercase, press the <CAPS LOCK> key. Press it again to type lowercase letters.

Step 8. Now enter the LOGOUT command as shown in Fig. 1-8.

The LOGOUT command gives you a final logout message, and then disconnects you from the VAX. It might also disconnect you from the system if your system requires a connect procedure. The "logout message" tells who was logged in and the date and time when that user logged out.

```
$ LOGOUT
  REAGANRON       logged out at 20-MAY-1987 13:09:56.79
```

Fig. 1-8. The screen after logging out.

DCL COMMANDS

Steps 3 through 8 constitute a complete "terminal session." These steps are essential to using the VAX, and should be repeated several times, so you can get used to logging in and out. This activity should become second-nature for you.

Step 9. Log in again, just as you did in Steps 3 and 4. In this step, you will execute a few more simple DCL commands. Appendix B lists many more. Several of these are "SHOW" commands; that is, they cause the system to report some information to you.

Enter the SHOW TIME command as shown in Fig. 1-9 (the space between the W and the T is required). The SHOW TIME command causes the current date and time to be displayed. These are obtained from the system clock, which is set when the system is first "brought up" (i.e., when the VAX is turned on).

```
$ SHOW TIME
  20-MAY-1987 13:19:56.82
```

Fig. 1-9. The screen as it appears when executing the SHOW TIME command. The VAX always has the correct time.

Next, enter the SHOW USERS command, which tells the operating system to display on your screen a list of all the other users who are currently logged in to the VAX. Of course, they are listed by their usernames, so you might not recognize some of them. If the screen display "scrolls" (i.e., moves up and off the screen) too fast, you can stop the scrolling by pressing the <NO SCROLL> key. Press it again to resume.

Step 10. When using the Digital Command Language (DCL), you might make a mistake when typing in a command, and not notice it. If you do, the system will respond with an error message to tell you why it couldn't understand you. For example, try the sequence shown in Fig. 1-10. Here, on the first try, the word "USERS" was misspelled. The DCL responded with the error code "%DCL-W-IVKEYW." The percent sign means "attention," the "DCL" means "Digital Command Language," the "W" means "Warning," and the "IVKEYW" stands for "INvalid KEYWord." The actual error message is "unrecognized keyword—check validity and spelling." Then the system points to the actual error by enclosing it in backslashes: "\USRS\."

On the second try, the same type of error was made, but this time the word "SHOW" was misspelled. Note that the first word in the SHOW USERS command is called the "command verb," while the second word is called a "keyword."

```
$ SHOW USRS
%DCL-W-IVKEYW, unrecognized keyword - check validity and spelling
 \USRS\
$ SHW USERS
%DCL-W-IVVERB, unrecognized command verb - check validity and
spelling
 \SHW\
$ SHOW US ERS
%SHOW-I-NOSUCHUSER, user ERS not currently logged in
$ SHOWUSERS
_What: <CTRL/Y>
 Interrupt
```

Fig. 1-10. The screen showing system responses to various user errors. The VAX is very particular about spelling.

On the third try, a blank space was accidentally put inside the word "USERS." Because "US" is an acceptable abbreviation for "USERS," this command was interpreted as "SHOW USERS ERS," which means to show if a user whose username is "ERS" is logged in. The DCL response code was "%SHOW-I-NOSUCHUSER", where the "I" stands for "Information." No warning was needed because the system was able to interpret the command; it was just the wrong interpretation. This illustrates several important aspects about DCL and the VAX system in general. First, the system is designed to always try to make a logical interpretation. As a result, it sometimes makes the wrong interpretation because of user error. Second, all commands may be abbreviated to their first few letters. For example, the SHOW USERS command may be abbreviated "SHO US". (See Appendix A for more information on abbreviations.)

On the fourth try, the user errs in the other direction: the necessary blank space between the command verb SHOW and the keyword USERS was omitted. The system interpreted the single word "SHOWUSERS" as meaning "SHOW," and then ignored the rest of the character string. Because the line buffer was sent to the VAX (by hitting the <RET> key) without including a separate keyword like "USERS," it replied by asking " What:" meaning, "What do you want me to show?" But this user didn't know what that reply meant, so he hit <CTRL/Y>.

The symbol "<CTRL/Y>" means to press the Y key while holding down the CONTROL key. This is like pressing the 5 key while holding down the SHIFT key to print the percent sign (%). The CONTROL/Y command is the "panic" command; use it when you don't know what else to do. It causes an *interrupt* of the current activity and returns you to the DCL Mode. You know that you are in DCL Mode when you see the cursor blinking next to the dollar sign prompt.

SELF-TEST EXERCISES

1. Practice logging in and logging out, i.e., repeat Step 8.
2. After logging in, enter some nonsense commands to see the different error messages that the system will give.
3. Try a few of the DCL commands listed in Appendix A.
4. Use the DCL HELP facility by following these steps:

   (a) Type HELP. This will cause a screen like Fig. 1-11 to appear. After you read the display, press <RET> to view the next screen of information. The VAX is telling you that it will provide information on over 100 different DCL commands. Continue pressing <RET> as long as you see the prompt, "Press RETURN to continue . . . ."

   (b) Type MAIL in response to the "Topic?" prompt. This will cause your screen to display information about the MAIL command.

   (c) Type EXAMPLES in response to the "MAIL Subtopic?" prompt. This will display several examples of how the MAIL command is used. Appendix E gives more information about the VAX's MAIL facility.

   (d) There are two ways to return to DCL Mode. One way is to continue pressing <RET> until you see the dollar sign prompt. This will cause the normal exit from each of the modes you have entered. The other way is to

```
$ HELP

HELP

   The HELP command invokes the VAX/VMS HELP Facility to display
   information about a VMS command or topic.  In response to the "Topic?"
   prompt, you can:

      o Type the name of the command or topic for which you need help.

      o Type INSTRUCTIONS for more detailed instructions on how to use
        HELP.

      o Type HINTS if you are not sure of the name of the command or
        topic for which you need help.


      o Type a question mark (?) to redisplay the most recently requested
        text.

      o Press the RETURN key one or more times to exit from HELP.


 Press RETURN to continue ...
```

Fig. 1-11. The screen as it appears when executing the HELP command. The VAX's extensive HELP files are one of its most attractive features.

```
$ HELP SET PASSWORD

SET

  PASSWORD

      Allows users to change their own passwords.  Also allows system
      managers to change the system password.

      Format:

        SET PASSWORD


    Additional information available:

    Command_Qualifiers
    /GENERATE  /SECONDARY  /SYSTEM
    Examples

SET PASSWORD Subtopic? /GENERATE
```

Fig. 1-12. The screen showing the system response to HELP SET PASSWORD. Nearly everything you need to know about a command is available on-line.

press <CTRL/Y>. This is the interrupt command which will always return you directly to DCL Mode.

5. Use the following steps to investigate the SET PASSWORD command with the HELP Facility:
   (a) Enter HELP SET PASSWORD as in Fig. 1-12.
   (b) Enter /GENERATE to learn how the VAX will generate a random set of passwords from which to choose.
   (c) Return to DCL Mode and enter SET PASSWORD/GENERATE to try out this facility. Be careful, however. The passwords which the VAX generates are cryptic enough to prevent anyone from guessing them, but, for the same reason, they are difficult to remember.

Chapter 2

The EDT Line Editor

For precept must be upon precept, precept upon precept; line upon line, line upon line; here a little, and there a little.

Isaiah 28:10

In Chapter 1, you learned how to execute some simple DCL commands on the VAX. The information which you typed at the keyboard was interpreted by the VAX's Command Interpreter as requests for the computer to take certain actions, such as telling you the time or changing your password. This is "real-time" activity. The information is used immediately, and then ignored by the computer. This is the kind of activity for which computers were originally invented: to carry out computations and report their results.

Modern computers, however, are also used for another very important purpose: the storage and retrieval of information. This information might consist of physical measurements, bank records, computer programs, or Caesar's Commentaries. Whatever it is, the computer stores it (usually on magnetic tape or disk) the same way. It is organized into **files,** analogous to the way information on paper is organized in an office.

THE FILE SPECIFICATION

Each file is identified by a unique name, called its *file specification,* or **filespec** for short. On the VAX, a filespec must conform to a rigid format. A

con ɔte filespec has six parts: a *node*, a *device*, a *directory*, a *filename*, a *type*, and a *version*.

Here is a typical VAX/VMS complete file specification:

VAX750::DBA0:[REAGANRON]TEST1.TXT;6

In this example, "VAX750" is the node, "DBA0" is the device, "REAGANRON" is the directory, "TEST1" is the filename, "TXT" is the type, and "6" is the version. Notice the required punctuation: the double colon (::) after the node, the single colon (:) after the device, the brackets ([]) enclosing the directory, the period (.) preceding the type, and the semicolon (;) preceding the version.

The node in a filespec is meaningful only if the VAX that accesses the file is part of a network of computers. In this case, the computer that controls the file's access is called its node. One of the nodes in the network at the University of Richmond is named "VAX750." If someone who has logged in on another node wants to access a file controlled by this node, the prefix "VAX750::" must be included in the filespec. If a user is already logged in on the VAX750 node, the prefix is unnecessary; the shorter filespec DBA0:[REAGANRON]TEST1.TXT;6 would suffice.

The device is the name of the storage device on which the file resides. In the example above, "DBA0" is the name of a magnetic disk device. For most users, this part of the filespec is omitted because they are allowed access to only one storage device on the computer. For this reason, the device will not be used any more in this book.

The directory in a filespec is the name of a catalog of files. You shall see, in Chapter 9, that this can have several parts. In simple cases, however, the directory is the same as the user's username. When this is the case, it, too, may be omitted by the user when referring to files in an account.

The filename is chosen by the user when the file is created. In the example above, the user chose the filename "TEST1," because the file was used to test the editor. A later test file might be named "TEST2." Notice that both letters and digits may be used in filenames. You may also use the dollar sign ($), the hyphen (-), and the underline (_). You may not, however, include any spaces. The underline is often used to simulate a space. For example, the string "THIS_IS_THE_FIRST_FILE_I_EVER_CREATED" could conceivably be used as a filename (up to 39 characters are allowed), but it is usually better to keep things simple.

The type (also called the *filetype* or *extension*) in a filespec is also chosen by the user when the file is created. In the example above, the type TXT stands for "TeXT file," which means that it is readable by humans. Other common choices are: BAS for "BASIC program" (see Chapter 7), COM for "command procedure" (see Chapter 9), DAT for "data file," DIR for "directory file" (see Chapter 8), EXE for "executable image" (Chapter 5), JOU for "journal file" (Chapter 2), MAI for "mail file" (Appendix I), OBJ for "object module" (Chapter 5), and PAS for "Pascal program" (Chapter 5). The restrictions on the kind and number of characters allowed for a file type are the same as for the filename. However, most

programmers stick to the standard three-letter types. If you need more expressive descriptions, put them in the filename.

The version in a filespec is set automatically by the VAX. When the file is created, it is given the version number "1." Then, whenever the file is revised or altered in any way, the result is a completely new file with the same filespec as its predecessor, except that its version number is increased by one. For example, if a file named ROOTS.PAS;27 is modified by the editor, the result will automatically be named ROOTS.PAS;28. Because the version of a file is handled automatically by the VAX, it is usually not referenced by the user.

CREATING AND REFERENCING FILES

You have seen that a complete filespec includes six independent parts. In most cases, however, only the filename and type are referred to by the user. The period (.) is the necessary punctuation separating these two parts. Here are some typical filespecs: MEMO.TXT for a memo to be read by someone else, DEPOSITS.DAT for a list of bank deposits, LOGIN.COM for a command procedure, MAIL.MAI for an indexed mail file, PROGRAM1.PAS for a Pascal program created with the editor, PROGRAM1.OBJ for the machine-language translation of the preceding Pascal program, PROGRAM1.EXE for the executable machine-language image file created from the OBJ file.

Suppose that you want to create a file in which to keep some notes to yourself. The simplest way to do this is to use the CREATE command.

Step 1. If you are not logged in on the VAX, do so now (Steps 3–7 in Chapter 1). Then, type in what is shown in Fig. 2-1. The CAPS LOCK key will allow lowercase letters to be typed. Do not try to use the cursor control keys (labeled with arrows) or the BACK SPACE key. If you make a mistake, use the DELETE key, and remember to press <RET> at the end of each line.

The filespec NOTE.TXT is given to the file being created. Each line typed after the CREATE command has been entered will be inserted into the file. In this example, the file will have three lines. The symbol <CTRL/Z> stands for the CONTROL/Z command, which is entered by pressing the Z key while holding down the CONTROL key (labeled "CTRL"). This tells the VAX: "End of File." Notice that the VAX echoes the CONTROL/Z command with "Exit" in reverse video, because it exits from the execution of the CREATE command.

Step 2. You have just created a file called NOTE.TXT. You can view its contents any time you are in DCL mode, simply by giving the TYPE command. Do this now, as shown in Fig. 2-2.

The TYPE command causes the contents of the specified file to be displayed. (The word "type" was originally chosen for this command in an era when nearly all output from the computer had to be typed by a printer.) Notice that the

```
$ CREATE NOTE.TXT
Call Lisa.
Cash check.
Buy milk.
<CTRL/Z>
```

Fig. 2-1. The screen as it appears when executing the CREATE command. Of the various ways to create a new file, the CREATE command is the simplest.

```
$ TYPE NOTE.TXT
Call Lisa.
Cash check.
Buy milk.
```

Fig. 2-2. The screen as it appears when executing the TYPE command. This displays the contents of the file.

file contains exactly what you typed between the CREATE command and the <CTRL/Z> command; nothing more, nothing less.

Step 3. The file that you have created will remain in your account until you delete it. To see that this is true, log off the VAX. Then, log in and do Step 2 again. The file is still there, unchanged.

The actual contents of a file are stored on a disk, in a disk pack which is separate from the VAX itself. But the location of that file on the disk, together with its filespec and other identifying data, are kept in your account on the VAX.

To see what is in your account, give the DIRECTORY command, as shown in Fig. 2-3.

```
$ DIRECTORY

Directory SYS$USER:[REAGANRON]

NOTE.TXT;1

Total of 1 file.
```

Fig. 2-3. The screen as it appears when executing the DIRECTORY command. This lists the names of all the files in your directory.

The DIRECTORY command causes the VAX to display the filespecs of all of the files in your account. Figure 2-3 shows only the new file, NOTE.TXT, in the account REAGANRON. (You might have more files already.) Notice that the file's version number is also displayed by the DIRECTORY command.

Step 4. Suppose that you want to add another reminder to your NOTE file. Unfortunately, the CREATE command is not flexible enough to allow this. You could create a new file and type in the contents of NOTE.TXT together with your new reminder, but there is a better way.

Changing a file on a computer is called "editing" the file. Most computer systems provide a subsystem, called an *editor*, to facilitate the editing of files. Smaller microcomputers, like the Commodore 64, come equipped with an editor which works automatically when the computer is turned on. Larger microcomputers, like the IBM PC, have editors which are loaded from a disk. The VAX has several editors. Its standard editor is called *EDT*.

The VAX's EDT editor itself has several subsystems. The simplest of these is its *Line Editing Mode*. In this mode, the editor manipulates the file one line at a time. Each line of the file is numbered by the editor as it is created. Then, the actions of the editor are carried out with reference to the line numbers.

Suppose that you want to remind yourself to pick up a package at the post

office before you cash your check. Type the commands shown in Fig. 2-4 to add this reminder to your NOTE file. Remember, <CTRL/Z> means to press the key labeled "CTRL" and then the "Z" key.

```
$ EDIT NOTE.TXT
    1       Call Lisa.
*TYPE WHOLE
    1       Call Lisa.
    2       Cash check.
    3       Buy milk.
[EOB]
*2
    2       Cash check
*INSERT
            Get package at post office.<CTRL/Z> Exit
    2       Cash check.
*TYPE WHOLE
    1       Call Lisa.
    1.1     Get package at post office.
    2       Cash check.
    3       Buy milk.
[EOB]
*EXIT
SYS$USER:[REAGANRON]NOTE.TXT;2  4 lines
```

Fig. 2-4. The screen showing the use of the EDT Line Editor and its TYPE WHOLE and INSERT commands. This is another way to create a new file.

The EDT Line Mode is invoked with the EDIT command. Because the file NOTE.TXT has already been created, the editor responds by displaying the first line of the file, together with its line number. Then it gives the EDT Line Mode prompt, which is the asterisk (*). Recall that a prompt is a symbol, which the system displays on the screen to serve two purposes: to identify the set of acceptable responses by the user, and to signal that the system is ready to receive the user's response. In this case, the asterisk prompt identifies the EDT Line Mode, and its appearance signals that the VAX is ready to accept an EDT Line Mode command.

The first EDT Line Mode command to be given here is the TYPE WHOLE command. This causes the editor to display the entire file. As before, each line is displayed with its line number. (The line numbers are not part of the file.) The symbol "[EOB]" is displayed after the last line of the file. This signifies the End Of the Buffer.

The next EDT Line Mode command given is the number 2, which tells the editor to display only line 2. It has another important effect which is not visible: it establishes line 2 of the *current line*. Recall that this mode is called a "line editor" because the sections of text to be manipulated are lines in the file. Consequently, in this mode, one line of the file is always designated as the current line. All actions which the editor takes are relative to that line. The simplest way to make a line the current line is to type its line number as an EDT Line Mode command.

Line 2 was made the current line because a new line is to be inserted directly above it. This insertion is done by giving the INSERT command, typing the

new line, and then typing <CTRL/Z>. Recall that <CTRL/Z> signals the end of the current mode, in this case, the end of the INSERT Mode. Notice that this new mode has no prompt symbol; instead, the cursor is positioned in column 13 on the next line. After <CTRL/Z> is pressed to exit from the INSERT Mode, the asterisk reappears, signalling that you are back in the EDT Line Mode.

Next, give the TYPE WHOLE command again to see the effect of the insertion. The new line was indeed inserted directly above line 2. Notice that the new line was given the line number 1.1. Insertions will not affect the line numbers of existing lines.

Finally, exit from the EDT Line Mode by giving the EXIT command. The editor will display the file's filespec and the current number of its lines and then return to the DCL Mode, signified by the dollar sign prompt. Notice that the complete filespec is given: SYS$USER:[REAGANRON]NOTE.TXT;2. The abbreviation SYS$USER is used in place of the node and device names.

Step 5. It was mentioned earlier that each time a file is modified, a new version of the file is created. To see that this has happened, enter the DIRECTORY command again.

THE INTERACTION OF MODES

When you first log in on the VAX, you are in the DCL Mode. The dollar sign prompt appears on the screen to remind you that you are in this mode, and to tell you that the VAX is ready to receive DCL commands. When you invoke the EDT Line Mode with the EDIT command, you enter the EDT Line Mode. The prompt for this mode is the asterisk (*). If you think of a computer operating mode as being like a room within which you are allowed to perform certain activities, then the VAX's EDT Line Mode is like being inside its DCL Mode. Like one room enclosing another, you must first enter the DCL Mode before you enter the EDT Line Mode. Similarly, when you exit from the EDT Line Mode, you are still inside the DCL Mode.

The room analogy is suggested in Fig. 2-5. Here, the larger room represents the DCL Mode, with one entrance in its left wall, and one exit in its right wall. The entrance is labeled USERNAME PASSWORD, and its exit is labeled LOGOUT. Inside the DCL Mode are located some of the DCL commands, including EDIT, SHOW TIME, and LOGOUT. The EDIT command is also the label of the door into the room which represents the EDT Line Mode. Inside this room are some of the EDT Line Mode commands, like INSERT.TYPE, and EXIT. The EDT Line Mode has two exits, one labeled EXIT, and one labeled QUIT. This means that you may type either EXIT or QUIT to leave the EDT Line Mode and return to the DCL Mode. If you use the QUIT command, your revised file will not be saved in your account.

Figure 2-5 illustrates another important point. There are some words, like DELETE and HELP, which serve as commands in more than one mode. The important point is that the meaning of the word depends upon the mode within which it is used. Just as the word "burger" has different meanings in English and German, so will the command "DELETE" have different effects in DCL Mode

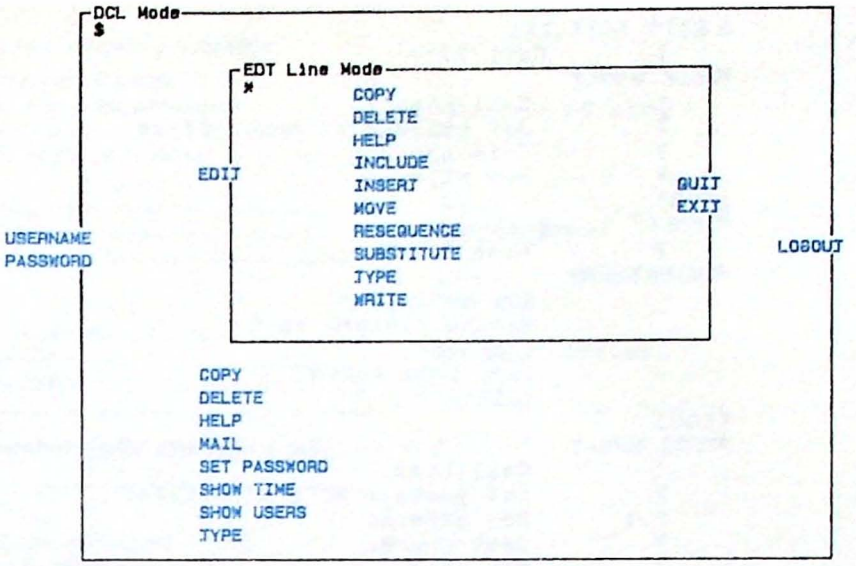

Fig. 2-5. The DCL Mode and the EDT Line Mode. The EDT Line Mode "lives inside" the DCL Mode.

and EDT Line Mode. Fortunately, you can always tell which mode you are in by the prompt.

More "rooms" will be added to Fig. 2-5 as you learn about more modes available within the VAX's DCL Mode. Appendix F discusses many of these modes.

Step 6. To learn more of the EDT Line Mode commands, you need a larger file. Follow the commands shown in Fig. 2-6 to add more lines to your NOTE.TXT file. Notice that, when you first enter the editor, the lines are renumbered as 1, 2, 3, etc., regardless of the line numbers from the previous version (but in the same order). Notice, also, the new use of the INSERT command. By following the command with a line number, a semicolon, and a new line of text, you can bypass the separate EDT INSERT Mode. The new line will immediately be added directly above the line whose number has been specified (line 3). This is quicker than entering a separate mode, but it works on only one new line at a time. The insertion is made when <RET> is pressed. The <CTRL/Z> is not needed.

The other four new lines are added at the end of the file, which is why you must type INSERT END. The word END serves as the line number for the End-Of-Buffer marker [EOB] after the last line of the file.

Step 7. Until now, the lines of a file have been referred to either explicitly by their line numbers, or with the word "END," which denotes the "line" after the last line in the file. The EDT Line Mode provides several other options.

Analogous to "END" is the word "BEGIN," which denotes the first line of the file. Initially, this would be the same as "1," because the first line is numbered 1 when the editor is invoked. Similarly, the period (.) denotes the

```
$ EDIT NOTE.TXT
    1       Call Lisa.
*TYPE WHOLE
    1       Call Lisa.
    2       Get package at post office.
    3       Cash check.
    4       Buy milk.
[EOB]
*INSERT 3;Buy stamps.
    3       Cash check.
*INSERT END
            Buy newspaper.
            Return library books.
            Wash car.
            Call Lisa again.
            <CTRL/Z>
[EOB]
*TYPE WHOLE
    1       Call Lisa.
    2       Get package at post office.
    2.1     Buy stamps.
    3       Cash check.
    4       Buy milk.
    5       Buy newspaper.
    6       Return library books.
    7       Wash car.
    8       Call Lisa again.
[EOB]
*EXIT
SYS$USER:[REAGANRON]NOTE.TXT;3 9 lines
```

Fig. 2-6. The screen when you are using the INSERT command to insert new lines into the existing file.

current line. If the current line is line 7, then typing "." would have the same effect as typing "7;" it would cause line 7 to be displayed.

The colon (:), plus sign (+), and negative sign (−) can be used with integers and with the constants "BEGIN," "END," and "." to refer to various parts of a file. For example, **.+5** refers to the line which is 5 lines after the current line, and **END−2** refers to the second to last line of the file. The colon (:) is used to designate a range of lines. For example, **28:51** refers to the block from line 28 to line 51, and **BEGIN:.+2** refers to the block from the beginning of the file to 2 lines past the current line.

Another useful way to refer to parts of a file is to type a string of letters enclosed in apostrophes ('). For example, entering **TYPE 'buy'** will locate the next line that contains the word "buy." This is particularly useful for searching through a file for a known string. If you want to search in the reverse direction (always from the current line), use a negative sign in front of the string. If you want to find all the lines which contain a given string, preface the string with the word "ALL." For example, **TYPE 3:END ALL 'ca'** will display all the lines from line 3 to the end of the file which contain the string "ca."

Follow the examples shown in Fig. 2-7 to get used to these reference codes for ranges in a file.

```
$ EDIT NOTE.TXT
     1        Call Lisa.
*TYPE WHOLE
     1        Call Lisa.
     2        Get package at post office.
     3        Buy stamps
     4        Cash check.
     5        Buy milk.
     6        Buy newspaper.
     7        Return library books.
     8        Wash car.
     9        Call Lisa again.
[EOB]
*<RET>
     2        Get package at post office.
*<RET>
     3        Buy stamps
*<RET>
     4        Cash check.
*.
     4        Cash check.
*.
     4        Cash check.
*.+2
     6        Buy newspaper.
*.
     6        Buy newspaper.
*3:5
     3        Buy stamps
     4        Cash check.
     5        Buy milk.
*.
     3        Buy stamps.
*.+3:END-2
     6        Buy newspaper.
     7        Return library books.
     8        Wash car.
*.
     6        Buy newspaper.
*'ca'
     8        Wash car.
*.
     8        Wash car.
*4:8 ALL 'buy'
     5        Buy milk.
     6        Buy newspaper.
*.
     4        Cash check.
*-'s.'
     3        Buy stamps.
*QUIT

$ DIRECTORY

Directory SYS$USER:[REAGANRON]

NOTE.TXT;3          NOTE.TXT;2          NOTE.TXT;1
Total of 3 files.
```

Fig. 2-7. The screen when you are using line codes and search codes to move around the editing buffer.

Note that simply pressing <RET> advances the current line one line at a time. Whenever the current line changes, it is displayed. After pressing <RET> three times, line 4 becomes the current line, and it is displayed. Entering a period, which causes the current line to be displayed, does not advance the current line. Entering **.+2** displays the line that is two lines past the current line, and makes that the current line. When a range like "3:5" is entered, those lines are displayed, and the first of them becomes the current line. Notice that the TYPE command is not needed when a range is specified.

Entering the character string "ca" causes the editor to search for the next line (starting with the current line) which contains that string. In Fig. 2-7, the next occurrence of "ca" is found in line 8, so it is displayed and becomes the current line. Next, all of the occurrences of the string "buy" within the range 4:8 are found. Notice that the current line then becomes line 4, not line 5. That is because line 4 is the first line in the given range. Also note that the search is done without regard to the case of letters specified. The string "Buy" is considered to be the same as the string "buy."

As mentioned earlier, the QUIT command terminates the current editing session without saving the file. There was no need to save the file (by entering EXIT instead of QUIT) in Fig. 2-7, because no changes were made during that editing session. This command is more efficient, because it is quicker and doesn't add a redundant file to your account. A quick check with the DIRECTORY command shows that you still have only the files that you had before.

Step 8. Suppose you decide that your business at the post office is not as important as your grocery shopping. To correct your note list, use the MOVE command, as shown in Fig. 2-8.

```
$ EDIT NOTE.TXT
    1       Call Lisa.
*TYPE WHOLE
    1       Call Lisa.
    2       Get package at post office.
    3       Buy stamps
    4       Cash check.
    5       Buy milk.
    6       Buy newspaper.
    7       Return library books.
    8       Wash car.
    9       Call Lisa again.
[EOB]
*MOVE 2:3 TO 7
2 lines moved
*TYPE WHOLE
    1       Call Lisa.
    4       Cash check.
    5       Buy milk.
    6       Buy newspaper.
    6.1     Get package at post office.
    6.2     Buy stamps
    7       Return library books.
    8       Wash car.
    9       Call Lisa again.
[EOB]*.
    1       Call Lisa.
```

Fig. 2-8. The screen when you are using the MOVE command to move several lines.

The MOVE command moves a range of lines to the position directly before the target line. Here, the range from line 2 to line 3 is moved to the target line 7. The only line numbers that change are those of the lines which are moved: line 2 becomes line 6.1, and line 3 becomes line 6.2. The MOVE command does not change the current line.

Step 9. Now suppose that you might not have time to get to the post office today, but you want to be sure to call Lisa back after you finish grocery shopping. So, you want to copy line 9 to the target line 6.1. Enter the commands shown in Fig. 2-9.

```
*COPY 9 TO 6.1
1 line copied
*TYPE WHOLE
     1        Call Lisa.
     4        Cash check.
     5        Buy milk.
     6        Buy newspaper.
     6.01     Call Lisa again.
     6.1      Get package at post office.
     6.2      Buy stamps
     7        Return library books.
     8        Wash car.
     9        Call Lisa again.
[EOB]
*.
     1        Call Lisa.
```

Fig. 2-9. The screen when you are using the COPY command to copy several lines.

The COPY command works like the MOVE command; the only difference is that the MOVE command deletes the range of lines from their original position. Notice that a range of lines might be made up of only one line. In this example, the range consists only of line 9. Note, also, how the editor selects its new line numbers. The new line to be inserted between line 6 and line 6.1 is given the line number 6.01. The line numbers are always in sequence.

Step 10. Suppose you decide that it will be impossible to get to the post office or the library, because you want to continue learning about the VAX. Enter the commands shown in Fig. 2-10. The DELETE command is similar to the MOVE and COPY commands. Note that the new current line, which is displayed automatically, is line 8, the first line after the deleted block.

By this time, the lines of your file have been jumbled up pretty well. The RESEQUENCE command renumbers the lines, simplifying their range specifications. It also resets line 1 as the current line.

Step 11. One of the most useful EDT Line Mode commands is the SUBSTITUTE command. This allows you to easily modify part of one or more lines in a single step. To see how it works, type the commands shown in Fig. 2-11.

The SUBSTITUTE command takes three *arguments*. An argument of a command is a modifier which provides the specific details of an action, so that the command interpreter can execute it. In the first use of the command, the

```
*DELETE 6.1:7
3 lines deleted
     8        Wash car.
*TYPE WHOLE
     1        Call Lisa.
     4        Cash check.
     5        Buy milk.
     6        Buy newspaper.
     6.01     Call Lisa again.
     8        Wash car.
     9        Call Lisa again.
[EOB]
*RESEQUENCE
7 lines resequenced
*.
     1        Call Lisa.
*TYPE WHOLE
     1        Call Lisa.
     2        Cash check.
     3        Buy milk.
     4        Buy newspaper.
     5        Call Lisa again.
     6        Wash car.
     7        Call Lisa again.
[EOB]
```

Fig. 2-10. The screen when you are using the RESEQUENCE command to renumber the lines. The lines are renumbered 1, 2, 3, . . .

```
*SUBSTITUTE/r/t/6
     6        Wash cat.
1 substitution
*SUBS/./;/1:6
     1        Call Lisa;
     2        Cash check;
     3        Buy milk;
     4        Buy newspaper;
     5        Call Lisa again;
     6        Wash cat;
6 substitutions
*S/sa/sa and Molly/1:'.'-1
     1        Call Lisa and Molly;
     5        Call Lisa and Molly again;
2 substitutions
*S/Lisa and //5
     5        Call Molly again;
1 substitution
*T W
     1        Call Lisa and Molly;
     2        Cash check;
     3        Buy milk;
     4        Buy newspaper;
     5        Call Molly again;
     6        Wash cat;
     7        Call Lisa again.
[EOB]
*EXIT
SYS$USER:[REAGANRON]NOTE.TXT;4 7 lines
```

Fig. 2-11. The screen when you are using the SUBSTITUTE command to replace a string of characters in a line.

arguments are r, t, and 6. Note that the "slash" character (/) precedes each argument. The meaning here is: "SUBSTITUTE for the string r the string t on line 6." After executing the command, the editor displays the modified line and reports that one substitution was made.

In the second example, every period (.) has been replaced by a semicolon (;) in the range from line 1 to line 6. Again, all of the modified lines are displayed, and the number of substitutions is reported. Notice in this example that the abbreviation SUBS is used in place of SUBSTITUTION. Nearly every command in every VAX mode has abbreviations like this. They are always the first few letters of the word; in other words, the abbreviations are truncations of the commands, chopping off the ends. The minimal number of first letters allowable for an abbreviation depends upon what the first few letters are.

Golden Rule: All commands in a given mode can be truncated to a certain minimum number of letters. The minimum number is the smallest number which precludes ambiguity within the mode, and is never more than 4.

The minimal number for the EDT Line Mode command SUBSTITUTE is one; that is, the first letter S, is a valid abbreviation for the command. So are SU, SUB, SUBS, SUBST, etc. The minimal number for the DCL command CONTINUE is three; that is, CON is the shortest abbreviation allowed for this command. This is because DCL also has the commands COBOL and COPY, so the truncation down to the first two letters would be ambiguous. Appendix D lists most of the EDT Line Mode commands. For each entry, the minimal abbreviation is indicated by **boldface** print. For example, the RESEQUENCE command is listed as **RES**EQUENCE, indicating that it may be truncated down to its first three letters. This convention is followed in most of the appendixes.

The third example in Fig. 2-11 is: "S/sa/sa and Molly/1:'.' – 1". This illustrates several things. First, the one-letter abbreviation, S, is used for the SUBSTITUTE command. Insertions into lines are also shown. The trick here is to choose a string of the size to locate the place where the new text is to be inserted. In this example, the string "sa" was a good choice because, among all the lines in the given range, it occurs only where the insertions are wanted. The string "isa" or even the string "Lisa" would also have worked. The string "a" would not work, however. Such a command would also have inserted the string "and Molly" in several other places. (Try it.) A third point worth noting about this example is the range specifier, "1:'.' – 1". This means "lines 1 through 6", because the first occurrence of the string '.' is in line 7.

The next command given in Fig. 2-11 is the TYPE WHOLE command. This is abbreviated T W.

Step 12. When you are in the EDT Line Mode, the VAX system maintains a *journal file,* which retains every keystroke you enter. If you exit the editor prematurely, your journal file is retained in your directory with the file type JOU. Exiting "prematurely" means that the editor is interrupted before you can give either the EXIT or the QUIT command. This would happen if a power failure shut down the VAX, or if you interrupted your session with <CTRL/Y>. The EXIT and the QUIT commands automatically delete the journal file.

The journal file is created so that you can recover your editing work, in case of an interruption, using the qualifier /RECOVER with the EDIT command. Follow the example in Fig. 2-12 to see how this works.

First, enter the EDT Line Mode to create a file named CAESAR.TXT. The abbreviation "ED" is used for the EDIT command. Usually, when you enter the EDT Line Mode, the first line in the file is made the current line and is displayed. This file does not yet exit, however, so that fact is displayed instead. Actually, the "current line" is the imaginary line designated by the [EOB] symbol.

```
$ ED CAESAR.TXT
Input file does not exist
[EOB]
*I;Friends, Romans, countrymen,
[EOB]
*T W
    1       Friends, Romans, countrymen,
[EOB]
*S/men,/men .../1
    1       Friends, Romans, countrymen ...
1 substitution
*<CTRL/Y> Interrupt

$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.JOU;1        NOTE.TXT;4          NOTE.TXT;3          NOTE.TXT;2
NOTE.TXT;1
Total of 5 files.
$ T CAESAR.JOU
I;Friends, Romans, countrymen,
T W
S/men,/men .../1
$ ED/REC CAESAR.TXT
Input file does not exits
I;Friends, Romans, countrymen,
T W
    1       Friends, Romans, countrymen,
[EOB]
S/men,/men .../1
    1       Friends, Romans, countrymen ...
    1       Friends, Romans, countrymen ...
*EX
SYS$USER:[REAGANRON]CAESAR.TXT;1 1 line

$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1        NOTE.TXT;4          NOTE.TXT;3          NOTE.TXT;2
NOTE.TXT;1

Total of 5 files.
$ T CAESAR.TXT
Friends, Romans, countrymen ...
```

Fig. 2-12. The screen when you are using the EDT command with the /RECOVER qualifier to restore the contents of the editing buffer after the session was aborted. The system will re-execute the EDT commands as they were stored in the JOU file.

Next, insert a single line, using the "direct" version of the INSERT command. Again, use the minimal abbreviation for the command: the single letter "I" in this case. Then, the TYPE WHOLE command, abbreviated T W, will show the complete file containing the single line. Use the SUBSTITUTE command, abbreviated by S, to make a slight modification of the file.

Technically, the lines of text that are currently being accessed in the editor do not yet form a file. While the editor is active, the lines reside in the *main buffer* of the editor. The editor also contains the given filespec, "CAESAR.TXT".

Now, interrupt the editor with the <CTRL/Y> command. This returns you to DCL Mode. The DIRECTORY command, abbreviated by "DIR" shows that the file CAESAR.TXT was never created. Instead, you have a new file named CAESAR.JOU. This is the journal file for the interrupted editing session. The journal file is really just a list of all the EDT Editor commands issued during that session. The TYPE command, abbreviated T, shows this. The existence of this journal file enables you to use the /RECOVER qualifier.

In the Digital Command Language (DCL), most commands can be modified with certain words, called *qualifiers.* Every qualifier must be preceded by a slash (/). One of the qualifiers which the EDIT command can take is the /RECOVER qualifier. When executed, the editor reads all of the commands from the journal file and re-executes them, just as it had done before the editing session was aborted. For this to work, the current directory must have a file with type JOU and a filename the same as that given with the EDIT/RECOVER command. In our example, the command is abbreviated ED/REC and the given filename is CAESAR. Notice that the editor displays all its activity as it reads commands from the journal file.

Now, exit the editor naturally, using the EXIT command, abbreviated EX. The DIRECTORY command will reveal that the journal file has been deleted and the desired file CAESAR.TXT finally exists. The TYPE command shows that it contains the right text.

SELF-TEST EXERCISES

1. Practice logging in, entering the EDT Line Mode, inserting text with the INSERT command, exiting the editor, and then logging out. Use both the INSERT Mode and the direct INSERT command for single lines. Use the DIRECTORY command to see the effects of your activities.
2. Use the EDT Editor to create a file with eight lines, line 1 being "This is line 1.," line 2 being "This is line 2.," etc. Then, practice the MOVE, COPY, DELETE, and RESEQUENCE commands in the file.
3. Name the combination of COPY and DELETE commands that would have the same effect as the single EDT command "MOVE 6:9 to 4".
4. Edit your file NOTE.TXT;3 (creating NOTE.TXT;5) and, using only the MOVE, COPY, DELETE, and SUBSTITUTE commands, make it identical (i.e., restore it) to NOTE.TXT;2.
5. Repeat Exercise 4, but restore NOTE.TXT;4 to NOTE.TXT;3.
6. Create a new file, called NEWFILE.TXT, and insert whatever text you like.

Then, practice using the MOVE, COPY, DELETE, SUBSTITUTE, TYPE WHOLE, RESEQUENCE, EXIT, and QUIT commands. Refer to Appendix B for the abbreviations for these commands.

7. Try to use the SUBSTITUTE command to replace the string "mi/hr" with "ft/sec." The editor would not accept the command SUBSTITUTE/mi/hr/ft/sec/. Why not? Fortunately, there is an alternative. You can use another ***delimiter*** in place of the slash (/), for example, the "pipe" (|): SUBSTITUTE|mi/hr|ft/sec|. Try some others.
8. In the EDT Line Mode, the apostrophe (') by itself serves as a repeat command. If you press this key and <RET>, the editor will repeat the last command executed. Try this out on your NOTE.TXT file. For example, repeat searching for the string "ll."
9. From the EDT editor, enter the HELP command. This is similar to the HELP command in DCL Mode. See what the utility says about the other commands that you have already used. For example, enter HELP SUBSTITUTE. Refer to Appendix B for some of the other commands.
10. The EDT commands INCLUDE and WRITE are quite useful for "cutting and pasting" parts of files. When in the EDT Mode, the command INCLUDE STUFF.TXT will cause the entire file STUFF.TXT to be copied into the file you are editing. The new lines will be inserted at the current line, unless you specify another range. Conversely, the command WRITE STUFF.TXT 23:44 would create a new file named STUFF.TXT and insert into it the block of lines 23 to 44 from the file you are editing. Use the HELP command to investigate these two commands, and then try them.

Chapter 3

Using The DCL to Manipulate Files

We need elucidation of the obvious more than investigation of the obscure.

Oliver Wendell Holmes

Now that you have learned how to create and edit files, the next step is to manipulate files in the DCL Mode. The VAX Digital Command Language (DCL) is the language of the VAX/VMS operating system. It consists of all of the commands which the VAX will understand when it is in its primary state, the DCL Mode. You have already learned some of these commands: EDIT, SHOW USERS, SHOW TIME, SET PASSWORD, LOGOUT, HELP and DIRECTORY.

THE DELETE AND PURGE COMMANDS

Each time you terminate the EDT Line Editor with its EXIT command, it creates a new version of the file being edited. Obviously, after a while, your account will become rather cluttered with old unwanted versions. One way to clear out your account is to use the DELETE command. Follow Fig. 3-1.

The DELETE command removes the file that you specify. Dont' use it unless you are certain that you don't want the file anymore; once a file is deleted, you cannot get it back. Note that this command requires the version number to be specified in the filespec. The minimal abbreviation for the DELETE command is DEL.

```
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1          NEWFILE.TXT;1          NOTE.TXT;6          NOTE.TXT;5
NOTE.TXT;4            NOTE.TXT;3             NOTE.TXT;2          NOTE.TXT;1

Total of 8 files.
$ DELETE NOTE.TXT;3
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1          NEWFILE.TXT;1          NOTE.TXT;6          NOTE.TXT;5
NOTE.TXT;4            NOTE.TXT;2             NOTE.TXT;1

Total of 7 files.
```

Fig. 3-1. The screen as it appears when executing the DELETE command. This will remove (forever) the file(s) specified.

If you have many versions of a file, you might want to remove more than one. The PURGE command removes all but the latest version of the file specified as shown in Fig. 3-2. This is one of the most frequently used commands in the DCL. The version number is not used with the PURGE command.

THE COPY AND APPEND COMMANDS

You have already seen that the words DELETE and HELP serve as commands in both the DCL Mode and the EDT Line Editing Mode, as does the word COPY. In the EDT Line Editing Mode, the elements that are manipulated are the lines of a file, and the COPY command in that mode allows you to duplicate these elements. Similarly, in the DCL Mode, the elements that are manipulated are files themselves. So, the COPY command in this mode allows you to duplicate complete files. Follow the example in Fig. 3-3 to see how this works.

The COPY command may be used with or without version numbers. This is

```
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1          NEWFILE.TXT;1          NOTE.TXT;6          NOTE.TXT;5
NOTE.TXT;4            NOTE.TXT;2             NOTE.TXT;1

Total of 7 files.
$ PURGE NOTE.TXT
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1          NEWFILE.TXT;1          NOTE.TXT;6

Total of 3 files.
```

Fig. 3-2. Executing the PURGE command. This will remove all but the last version of the file(s) specified.

```
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1          NEWFILE.TXT;1          NOTE.TXT;6

Total of 3 files.
$ COPY NOTE.TXT NOTE.BAK
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;1          NEWFILE.TXT;1          NOTE.BAK;1          NOTE.TXT;6

Total of 4 files.
```

Fig. 3-3. Executing the COPY command. This will create an exact duplicate of the file specified.

typical of many of the DCL commands. If you omit the version number, the latest version is used.

The APPEND command is similar to the COPY command. See Fig. 3-4. Notice that the APPEND command does not create a new version of the object file (Fig. 3-5).

The APPEND command is one way to combine files, used when the resultant file already exists. If the filename of the resultant file does not need to be the same as one of the original files, then the more flexible COPY command should be used. Two or more files can be combined by using the COPY command with the

```
$ T CAESAR.TXT
Friends, Romans, countrymen ...
Lend me your ears!
$ CREATE C2.TXT
I come to bury Caesar,
Not to praise him.
<CTRL/Z>
$ DIR

Directory SYS$USER:[REAGANRON]

C2.TXT;1              CAESAR.TXT;2           NOTE.BAK;1          NOTE.TXT;6

Total of 4 files.
$ APPEND C2.TXT CAESAR.TXT
$ DIR

Directory SYS$USER:[REAGANRON]

C2.TXT;1              CAESAR.TXT;2           NOTE.BAK;1          NOTE.TXT;6

Total of 4 files.
$ T CAESAR.TXT
Friends, Romans, countrymen ...
Lend me your ears!
I come to bury Caesar,
Not to praise him.
```

Fig. 3-4. Executing the APPEND command. This will put a copy of the first file specified at the end of the second file specified.

```
$ APPEND FILE2.TXT FILE1.TXT
```

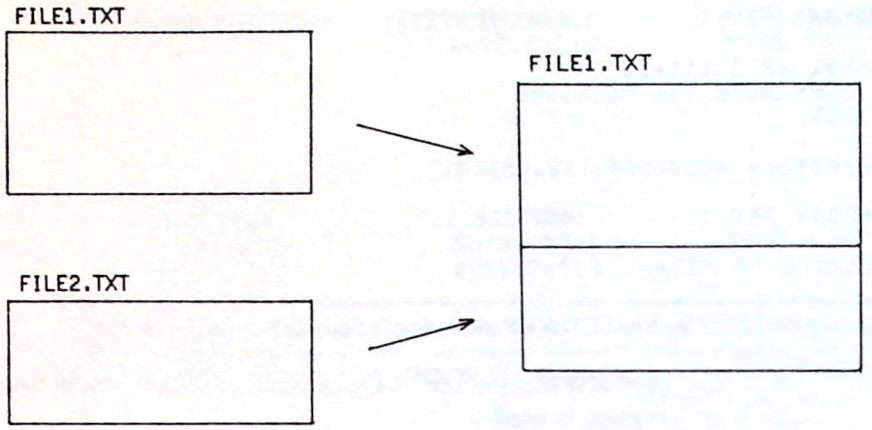

Fig. 3-5. Using the APPEND command to combine two files. The resulting file replaces the first file.

comma (,) between them. Follow Fig. 3-6 to combine four copies of your NOTE.TXT file into the new file N.TXT.

Concatenation

Of course, in this case, the resulting file N.TXT is a useless concatenation of four identical copies of the NOTE.TXT file. A more realistic example would be the concatenation of several different files to make up a single composite file. Large programs are often written this way. Their individual parts are written separately and then combined (Fig. 3-7).

```
$ COPY NOTE.TXT,NOTE.TXT,NOTE.TXT,NOTE.TXT N.TXT
$ DIR

Directory SYS$USER:[REAGANRON]

C2.TXT;1            CAESAR.TXT;2        N.TXT;1          NOTE.BAK;1
NOTE.TXT;6

Total of 5 files.
$ DEL C2.TXT;1, NOTE.BAK;
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2        N.TXT;1             NOTE.TXT;6

Total of 3 files.
```

Fig. 3-6. Executing the COPY command on several files. This is a common way to concatenate several files.

```
$ COPY FILE1.TXT, FILE2.TXT FILE3.TXT
```

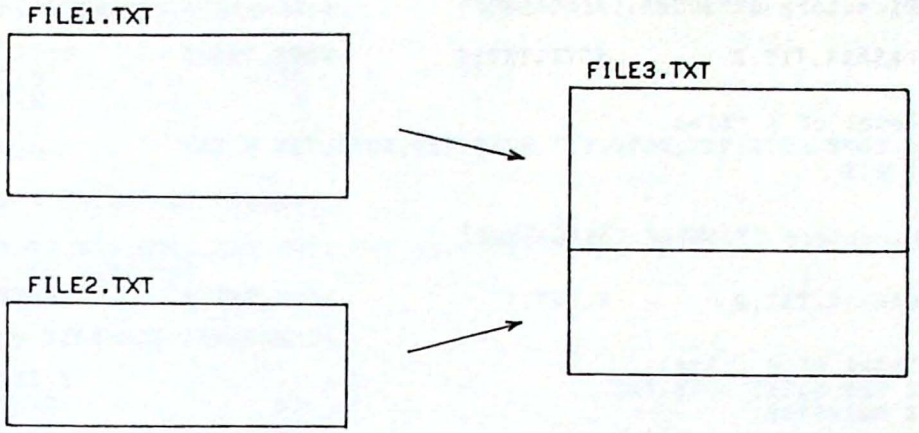

Fig. 3-7. Using the COPY command to combine two files. The resulting file is a new file.

The DELETE command was also used with the comma, to combine the effect of the command. This may be done with many DCL commands, including the PURGE and APPEND commands. The new file N.TXT that was just created is four times as large as the NOTE.TXT file. Enter TYPE N.TXT to see that it has 28 lines. Then follow the example in Fig. 3-8.

THE DIRECTORY/SIZE COMMAND

After renaming N.TXT, the concatenated COPY command is used again to produce a new file with 112 (4×28) lines. The DIRECTORY/SIZE command reveals how large these files are.

The /SIZE qualifier used with the DIRECTORY command causes each filespec to be listed with its size in blocks. On the VAX, one **block** equals 512 bytes. This means that each block of computer memory can hold 512 characters (letters, digits, punctuation symbols, etc.). The file NOTE.TXT;8 takes up 4 blocks of memory, so it has at most 2048 (4×512) characters in it. You know that it contains 112 lines, so it follows that each line has an average of fewer than 19 characters ($112 \times 19 > 2048$).

Managing Memory

As you create larger and larger files, it becomes more important to keep track of how much memory they take up. This is because the total amount of memory at your disposal is limited. The limit on the memory available to each account is established by the VAX's system manager when the account is created. This limit is called the *quota* for the account. The SHOW QUOTA command will tell you the limit on your account (Fig. 3-9).

The account in Fig. 3-9 has been given a total of 400 blocks of memory to

```
$ REN N.TXT NOTE.TXT
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2        NOTE.TXT;7        NOTE.TXT;6

Total of 3 files.
$ COPY NOTE.TXT,NOTE.TXT,NOTE.TXT,NOTE.TXT N.TXT
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2        N.TXT;1           NOTE.TXT.7        NOTE.TXT;6

Total of 4 files.
$ REN N.TXT NOTE.TXT
$ DIR/SIZE

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2              1
NOTE.TXT;8                4
NOTE.TXT;7                1
NOTE.TXT;6                1

Total of 4 files, 7 blocks.
```

Fig. 3-8. Executing the DIRECTORY command with the /SIZE qualifier. This tells how many blocks each file occupies. One block equals 512 bytes (characters).

use. This is 204,800 bytes (400 blocks × 512 bytes/block) or 200 kilobytes, which is about the size of an ordinary floppy diskette used in a microcomputer. Notice that 25 blocks have been used already, although all of the files take up only seven blocks. The other 18 blocks are used for overhead, and are beyond the control of the user. The important number in this report is 375. That is how much memory the user has left.

```
$ SHOW QUOTA
  User [STUDENT,REAGANRON] has 25 blocks used, 375 available,
  of 400 authorized and permitted overdraft of 100 blocks on SYS$USER
```

Fig. 3-9. Executing the SHOW QUOTA command. If you use up your quota, you may get stuck in the editor.

To see what happens when a user runs out of memory, follow the example in Fig. 3-10. This requires you to repeat the COPY, RENAME, and DIRECTORY commands several times. An easy way to do this is to use the RECALL command. This is analogous to the repeat command in the EDT Line Editor Mode,

```
$ COPY NOTE.TXT,NOTE.TXT,NOTE.TXT,NOTE.TXT N.TXT
$ REN N.TXT NOTE.TXT
$ DIR/SIZE

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2                1
NOTE.TXT;9                 15
NOTE.TXT;8                  4
NOTE.TXT;7                  1
NOTE.TXT;6                  1

Total of 4 files, 22 blocks.

$ COPY NOTE.TXT,NOTE.TXT,NOTE.TXT,NOTE.TXT N.TXT
$ REN N.TXT NOTE.TXT
$ DIR/SIZE

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2                1
NOTE.TXT;10                57
NOTE.TXT;9                 15
NOTE.TXT;8                  4
NOTE.TXT;7                  1
NOTE.TXT;6                  1

Total of 4 files, 79 blocks.

$ COPY NOTE.TXT,NOTE.TXT,NOTE.TXT,NOTE.TXT N.TXT
$ REN N.TXT NOTE.TXT
$ DIR/SIZE

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2                1
NOTE.TXT;11               228
NOTE.TXT;10                57
NOTE.TXT;9                 15
NOTE.TXT;8                  4
NOTE.TXT;7                  1
NOTE.TXT;6                  1

Total of 4 files, 307 blocks.
$ COPY NOTE.TXT,NOTE.TXT,NOTE.TXT,NOTE.TXT N.TXT
%COPY-E-OPENOUT, error opening SYS$USER:[REAGANRON]N.TXT; as output
-RMS-E-CRE, ACP file create failed
-SYSTEM-F-EXDISKQUOTA, disk quota exceeded
%COPY-W-NOTCOPIED, SYS$USER:[REAGANRON]NOTE.TXT;11 not copied
%COPY-W-NOTCOPIED, SYS$USER:[REAGANRON]NOTE.TXT;11 not copied
$ DIR/SIZE

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2                1

NOTE.TXT;11               228
NOTE.TXT;10                57
NOTE.TXT;9                 15
NOTE.TXT;8                  4
NOTE.TXT;7                  1
NOTE.TXT;6                  1

Total of 4 files, 307 blocks.
```

Fig. 3-10. The screen as it appears when you exceed your memory quota. When this happens, the VAX will not execute any more commands that require more memory.

which is executed with the apostrophe character (see Self-Test Exercise 8 in Chapter 2). The RECALL command is most easily executed on DEC terminals by just pressing the "up-arrow" cursor control key. If your keyboard does not have cursor control keys then enter the "REC" abbreviation for the RECALL command. In this example, you are to repeat the command executed three commands earlier. So, each time, press the up-arrow key three times to recall the correct command, and then enter it with <RET>. Notice that as the files get larger, it takes longer to copy them.

On the fourth attempt (or later, if your quota is greater) quadrupling the size of the NOTE.TXT file causes the execution of the COPY command to fail. The VAX responds with several error messages. The first one is abbreviated %COPY-E-OPENOUT. The percent character (%) signals that this is an error message. The word COPY refers to the command which failed to execute. The character "E" stands for ERROR, and the string "OPENOUT" is an abbreviation for the message "error opening . . . as output," which is also printed. The third line of the response tells why the error occurred, "disk quota exceeded," abbreviated with the code EXDISKQUOTA. The last two lines are "WARNINGS" (coded with W) to advise you that something you might have expected to occur did not. The last call to DIR/SIZ shows that the last COPY command was unsuccessful.

DCL WILDCARDS

Like most good operating systems, the VAX/VMS system allows the use of *wildcards*. These are symbols which can be used in place of other symbols. The VAX DCL has two wildcard characters: the asterisk (*), and the percent sign (%).

To see how these work, follow the example in Fig. 3-11. First, two more files, NAMES.DAT and NULL.PAS, are created just to have something to work with. Then, the DIRECTORY command is used in various ways. The string "*A*" stands for all filenames which include at least one "A," and the string "*.*;1" includes all filespecs whose version number is 1. If the version number is omitted, the system interprets the omission as meaning all versions. Thus, NOTE.TXT is equivalent to NOTE.TXT;* in the DIRECTORY command.

The asterisk wildcard represents any number (zero or more) of any characters. For example, in the exercise above, "*A*" represented both CAESAR and NAMES. The percent wildcard represents just one character. Thus, "%A*" would represent both CAESAR and NAME, but "*A%" would exclude NAMES, because its penultimate character is not an A.

Wildcards may be used with various DCL commands. For example, the command DEL *.DAT; deletes the latest version of all files whose type is DAT, and REN NOTE.TXT;6 *.*;12 changes the version of NOTE.TEXT from 6 to 12. With the RENAME command, the asterisks in the second filespec represent the same strings as are given in the corresponding parts of the first filespec. Using the RENAME command with the PURGE command in this example would provide a quick way to delete all but version 6 of the NOTE.TXT file.

DCL wildcards are powerful, and, therefore, can be dangerous, especially when used with the DELETE command. In particular, you are advised NOT to

```
$ CREATE NAMES.DAT
Molly Malone
Lisa Lassan
<CTRL/Z>
$ CR NULL.PAS
program X;begin end.
<CTRL/Z>
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2          NAMES.DAT;1          NOTE.TXT;11          NOTE.TXT;10
NOTE.TXT;9            NOTE.TXT;8           NOTE.TXT;7           NOTE.TXT;6
NULL.PAS;1

Total of 9 files.
$ DIR *A*.*

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2          NAMES.DAT;1

Total of 2 files.

$ DIR *.*;1

Directory SYS$USER:[REAGANRON]

NAMES.DAT;1           NULL.PAS;1

Total of 2 files.

$ DIR NOTE.TXT

Directory SYS$USER:[REAGANRON]

NOTE.TXT;11          NOTE.TXT;10          NOTE.TXT;9           NOTE.TXT;8
NOTE.TXT;7           NOTE.TXT;6

Total of 6 files.
$ DIR *A%.*

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2          NAMES.DAT;1

Total of 2 files.

$ DIR %A*.*

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2

Total of 1 file.
$ DEL *.DAT;
$ REN NOTE.TXT;6 *.*;12
$ PUR *.TXT
$ DIR

Directory SYS$USER:[REAGANRON]

CAESAR.TXT;2          NOTE.TXT;12           NULL.PAS

Total of 3 files.
```

Fig. 3-11. Using the % and * wildcards. These allow more efficient use of many DCL commands.

enter the command DEL *.*;*. Remember that once a file has been deleted, it cannot be recovered.

Successful computer systems, like the VAX/VMS Digital Control Language (DCL) and the EDT Line Editor, are continually being improved by their creators. Each major revision is labeled with a new version number. The current version of DEC's VAX/VMS operating system, at this writing, is called "Version 4."

DEFINING KEYPAD KEYS

One of the nicer features added in Version 4 of VAX/VMS is the ability to define the keypad keys so that they can be used to enter DCL commands. Recall that the keypad on the DEC VT100 and VT200 series keyboards is the block of 18 keys at the far right. The top four keypad keys are labeled PF1 through PF4. In the EDT editor, the keypad keys can also be used for full-screen editing; this will be discussed in Chapter 4. In DCL Mode, the keypad is governed by two possible modes: the Numeric Keypad Mode and the Application Keypad Mode. The default is the Numeric Keypad Mode. On the keypad, the period (.) key, the comma (,) key, the minus (—) key, and the ten digit keys (labeled 0 through 9) ordinarily have the same effect as those keys with the same labels on the main keyboard, and the ENTER key and the four PF keys have the same effect as the RETURN key. If you change your terminal to the Application Keypad Mode with the SET TERMINAL/APPLICATION_KEYPAD command, however, you can "customize" your keypad.

As an example, you will define the first three PF keys on the keypad. The PF1 key will be defined to enter the DIRECTORY/SIZE command, the PF2 key will enter the SHOW USERS command, and the PF3 key will enter the SHOW QUOTA command. Follow Fig. 3-12 to make these modifications.

Abbreviated Commands

The string SET TE/AP is the minimal abbreviation for the SET TERMINAL APPLICATION_KEYPAD command, and the string DEF/K/T is the minimal abbreviation for the DEFINE/KEY/TERMINATE command. Here, the /TERMINATE qualifier means that the command will be sent to the VAX as soon

```
$ SET TE/AP
$ DEF/K/T PF1 "DIRECTORY/SIZE"
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$ DEF/K/T PF2 "SHOW USERS"
%DCL-I-DEFKEY, DEFAULT key PF2 has been defined
$ DEF/K/T PF3 "SHOW QUOTA"
%DCL-I-DEFKEY, DEFAULT key PF3 has been defined
```

Fig. 3-12. Executing the DEFINE/KEY and the DEFINE/KEY/TERMINATE commands. In addition to saving time, this also saves "wear and tear" on your fingers and on the keyboard.

as the key is pressed; nothing more is to be added to it, so it can be terminated. Without the /TERMINATE qualifier, the command will be displayed on the screen when the key is pressed, but the user must press <RET> to send the command to the VAX. This allows the user to add more to the command—another qualifier, for example—before terminating it. Notice that the VAX responds to each DEFINE/KEY command with a %DCL message. Here, the "I" stands for INFORMATIONAL, and the "DEFKEY" stands for DEFINE/KEY.

After completing these definitions, try them out by pressing the PF keys.

SELF-TEST EXERCISES

1. What combination of COPY and DELETE commands would have the same effect as the single DCL command RENAME T1.TXT T2.TXT?
2. What combination of APPEND commands would have the same effect as the command COPY F1.TXT,F2.TXT,F3.TXT,F4.TXT F.TXT?
3. Use the HELP facility to learn more about the DCL commands which you have learned: APPEND, COPY, CREATE, DELETE, DIRECTORY, EDIT, HELP, LOGOUT, MAIL, PURGE, RENAME, SET PASSWORD, SHOW QUOTA, SHOW TIME, SHOW USERS, and TYPE. Pay particular attention to the various parameters and qualifiers.
4. The DIRECTORY command is one of the most flexible and frequently used commands in the DCL. Enter HELP DIR to see the 27 qualifiers that can be used with this command. Then enter "Examples" to see how some of these are used. Try them out yourself. In particular, try to decipher the information provided by the DIR/FULL command.
5. Investigate the DIRECTORY/EXCLUDE command. First, see what HELP has to say about it. Then, see what the commands DIR/EXC=*.TXT and DIR/EXC=(C*.*,NULL.PAS) do. In the second example, the two filespecs listed in parentheses are parameters for the qualifier EXCLUDE. The effect is to exclude the file NULL.PAS, and all files of the form C*.*, from the directory listing. In general, if a qualifier takes a parameter, the equal sign (=) is used to designate the parameter value (as in the first example here). If several parameter values are to be listed (as in the second example), the list must be enclosed in parentheses.
6. Investigate the CONTINUE command. It is used to resume the execution of any command which is interrupted by <CTRL/Y>. Try it after interrupting the SHOW USERS or DIRECTORY/FULL command.
7. If you are a student expecting to hand in programming assignments, you probably will have to insert a standard identification header at the beginning of each file. For example, at the University of Richmond, every Pascal program submitted for grading must begin with a three-line header in the following form:

```
(*****************************************************************)
(*  R. Reagan              CS201/JRH              Sept. 9, 1986  *)
(*****************************************************************)
```

Create a file called HEADER.TXT which contains such a header. Then,

whenever you begin a new file which requires this header, you can just COPY it to the new file, before you use the editor. Alternatively, you could write the new file first, and then use the INCLUDE HEADER.TXT command while you are in the editor (see Self-Test Exercise 10 in Chapter 2).

8. One way to discover the minimal abbreviation for a given command (in DCL or in EDT) is to experiment. (If you truncate it too much, you'll just get an error message.) For example, suppose you are investigating the SHOW LOGICAL/SYSTEM SY command. First, enter the command using all 22 characters shown here. Then, use the up-arrow cursor control key (or enter RECALL) to retype the complete command. Before pressing <RET> to execute it again, however, move the cursor left (using the left-arrow key) to the "H" and type 3 spaces over the "HOW" in the word SHOW. (Do not use the DELETE key.) Then, press <RET> to see if the one-letter truncation, S, is an acceptable abbreviation for SHOW. (It isn't.) Now, hit the up-arrow key twice, and, this time, just blank out the last 2 letters of the word SHOW. Continue this process until you find the minimal abbreviation for the SHOW. Then, do the same for the words LOGICAL and SYSTEM. This trial-and-error process seems complicated, but once you've done it, it becomes very quick and easy. Moreover, the process can be used with many other sub-systems on the VAX.

Chapter 4

The EDT Keypad Editor

One must learn by doing the thing; for though you think you know it, you have no certainty until you try.

Sophocles

Chapter 2 described how to use the EDT editor in its Line Mode. Nearly every computer operating system comes equipped with a line editor like EDT, in which the blocks of text are treated as sequences of lines. Line editing, however, can be a bit cumbersome. Often, it would be more convenient to be able to access the individual letters of a line directly, the way you can with a typewriter. This facility is provided by *character* editors.

A character editor, also called a *full-screen* editor, is similar to a word processor. It treats a block of text as a sequence of individual characters, allowing access to each character individually, regardless of which line it is on. The cursor always points to the current character, and all activity is with reference to that point in the editing buffer. The computer screen shows you exactly what is in the buffer; what you see is what you get. No line numbers or other control markings are visible.

THE KEYPAD MODE

The VAX imbeds its character editor within the EDT Line Mode. The submode which embodies its character editor is called the *Keypad Mode.* This name reflects the fact that the character editing commands are executed by

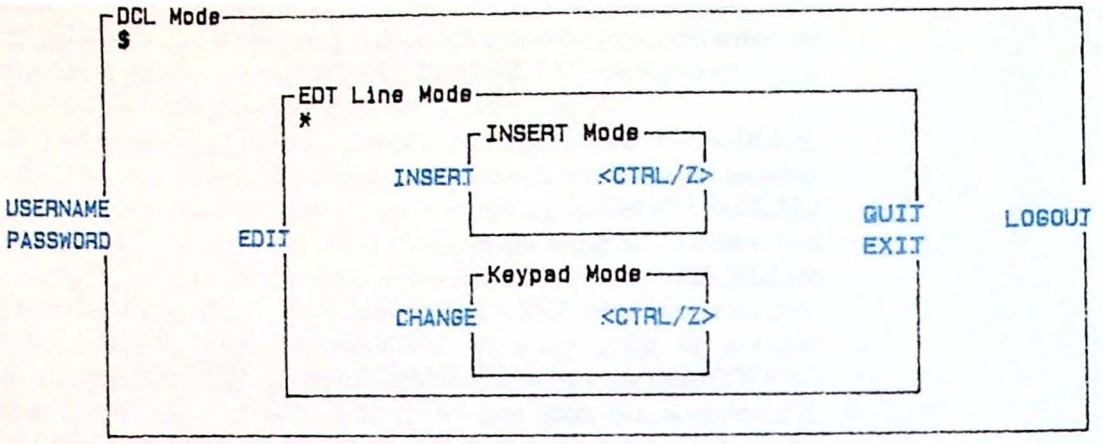

Fig. 4-1. The DCL Mode, the EDT Line Mode, the INSERT Mode, and the Keypad Mode. The INSERT and Keypad Modes "live inside" the EDT Line Mode.

means of the keypad keys to the right of the keyboard. (This mode is also sometimes called EDT's *CHANGE Mode*, because the EDT command CHANGE invokes it.) The relationship of the Keypad Mode to other modes is shown in Fig. 4-1.

To get into the Keypad Editor, first invoke the Line Editor. In this example, you will edit a letter, so call the file LETTER.TXT. See Fig. 4-2.

```
$ ED LETTER.TXT
Input file does not exist
[EOB]
*C
```

Fig. 4-2. Executing the CHANGE command in the EDT Line Mode. This "changes" the editor into a full-screen editor, allowing you to use the cursor control (arrow) keys.

Recall what happens when the EDT editor is invoked. It opens several text buffers to store ASCII characters. The default buffer is named MAIN, which is where the characters are inserted unless you specify another buffer. Another buffer which is created automatically is called PASTE, which is used for the "cut-and-paste" commands.

Notice that EDT will tell you, "Input file does not exist." This simply means that you don't already have a file named LETTER.TXT in your account. That file will not be created until you EXIT the editor. However, the editor does create another file as soon as it is invoked. That is the journal file, which holds every EDT command that you execute during the editing session. If the session should be interrupted, by pressing <CTRL/Y>, for example, you will find the file LETTER.JOU in your directory. (See Step 12 in Chapter 2.)

Give the CHANGE command, abbreviated by C. This changes the mode to Keypad Mode. It will clear the screen and then display only the end-of-buffer marker: [EOB].

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with computers since you
worked with Charles on his Analytical Engine.  Your predictions about their
usefulness have all come true!

     You were absolutely right about how well computers are able to process
textual information.  Indeed, I am using a computer now to write this letter to
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]
```

Fig. 4-3. Writing a letter in the EDT Keypad Mode. This works much like a word processor.

Type the letter shown in Fig. 4-3 into the editing buffer. Remember that releasing the <CAPS LOCK> key will allow you to type in lowercase. Use only the ordinary keyboard keys to type this letter. Do not try to use the <BACK SPACE> key; instead, use the <DEL> key to correct errors. Use the cursor control keys (marked with arrows) if you need to go back to correct anything, and use the <RET> key to end each line. If you type more than 80 characters on a line, a little diamond-shaped marker will appear in place of the 80th character. To fix this, just back the cursor to the left (using the left-arrow key) until you can see the cursor again, then press <RET> to begin a new line.

When you have finished typing the letter, press <CTRL/Z>. This returns you to the EDT Line Mode, as indicated by the asterisk prompt. Then, give the modified TYPE WHOLE command, as shown in Fig. 4-4. The qualifier /BRIEF:67 (/B:67) tells EDT to truncate each line to 67 characters. Because the line numbers take up 13 columns on the left of the screen, displaying any

```
*T W/B:67
    1
July 22, 1986
    2
    3       Dear Ada,
    4
    5            You would truly be amazed at the progress made with computers
    6       worked with Charles on his Analytical Engine.  Your predictions abo
    7       usefulness have all come true!
    8
    9            You were absolutely right about how well computers are able to
   10       textual information.  Indeed, I am using a computer now to write this
   11       you.  It was made by some very clever people here in Massachusetts.
   12
   13                                                    Your humble serva
   14
   15                                                    Herman
[EOB]
```

Fig. 4-4. Executing the TYPE WHOLE command with the /BRIEF qualifier. This prevents "wrap-around" by truncating each line.

more than 67 characters would cause the lines to be "wrapped around," making the buffer more difficult to review.

Reenter the Keypad Mode with the CHANGE command. The buffer will show exactly what is in your file. The cursor will be in the first column of the first line.

Practice moving the cursor around the screen using the cursor control keys (labeled with arrows). Their use will not affect the contents of your buffer in any way. The cursor simply serves as a locator. Notice that if you move the cursor past either end of a line, it just wraps around to the neighboring line. On the other hand, if you go too far up or down, you get an error message on the bottom line in reverse video. Also note that the cursor controls have an **auto-repeat** feature; if you hold the key down for more than a second, the command controlled by that key repeats rapidly until you release it. Most of the keys on the keyboard will behave this way.

In the EDT Line Mode, there is always one line singled out as being the current line. In the Keypad Mode, there is always one current line, one current word, and one current character. These are the line, the word, and the character under the cursor. The cursor always points to where the action is. Therefore, it is essential to know how to move the cursor around efficiently.

The Keypad Map

You have already seen one way to move the cursor around—just use the four cursor control keys. There are other ways which are often more efficient, but which require an understanding of the DEC VT100 *Keypad Map*, shown in Fig. 4-5.

The original purpose of the keypad, also called the "numeric keypad," was to facilitate the rapid entry of large amounts of numerical data. Figure 4-5 shows this keypad with each key label in its upper left corner. Consistent with the <RET> and <CTRL/Z> notation, each keypad key will be denoted by its label enclosed in angle brackets. For example, <PF1> denotes the upper left keypad key, and <0> denotes the lower left keypad key. The DEC VT100 Keypad Map also shows one or two other labels for each key. For example, the <6> key has the labels CUT and PASTE. These labels indicate the functions which the keys perform when you are in the Keypad Mode of the EDT editor.

The Keypad Map (Fig. 4-5) indicates that the <PF2> key executes the HELP command. Press that key now to see how the HELP command works in the Keypad Mode. The screen display summarizes essentially every Keypad Mode command available. Notice that the Keypad Map is shown on the left half of the screen. You will probably want to access this screen frequently, especially if you do not have a hard copy of the VT100 Keypad Map. (Some computer centers tape a copy on the front of each terminal.) Press the spacebar to return to the Keypad Mode.

CHARACTER EDITOR COMMANDS

Press the <1> key a few times. This key executes the WORD command,

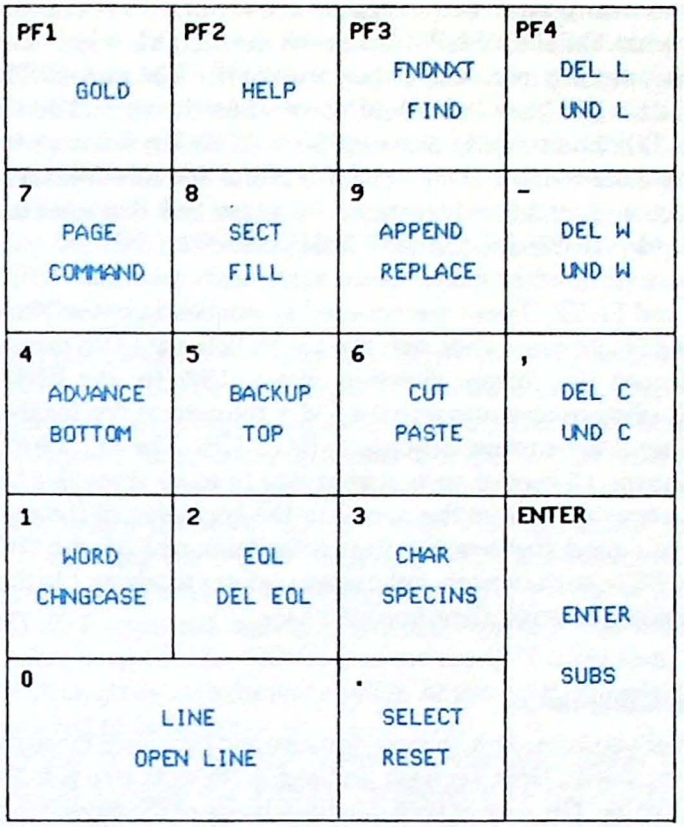

Fig. 4-5. The DEC VT100 (and VT200) Keypad Map. This map also appears on the back cover of this book. It may be taped to your terminal.

which moves the cursor to the beginning of the next word, the same way as the left-arrow cursor control key advances by letters. Hold the <1> key down for a few seconds: the auto-repeat of the WORD command is a quick way to advance through the buffer.

Press the <5> key to execute the BACKUP command. This reverses the current direction. Now press the <1> key a few more times; the WORD command still moves the cursor to the beginning of the next word, but now "next" means back—to the left and up. To reset the current direction to forward (i.e., to the right and down), execute the ADVANCE command by pressing the <4> key. Check the effect with the WORD command. You will encounter several Keypad Mode commands which depend upon the current direction (controlled by the <4> and <5> keys).

Next, execute the LINE command by pressing the <0> key. Try it with the BACKUP and ADVANCE commands. It works just like the WORD command, except that it moves to the beginning of the next line.

You can move the cursor to the top or to the bottom of the buffer by using any of the commands described above, but the best way is simply to use the TOP and BOTTOM commands. These words are the lower labels on the <5> and <4> keys. The lower label command on each keypad key is executed by press-

ing the <PF1> key first. This is called the "GOLD" key in DEC-land. The GOLD key works like the <SHIFT> key and the <CTRL> key: it changes the effect of the next key pressed. Unlike the <SHIFT> and <CTRL> keys, however, it does not have to be held down while the second key is pressed. Execute the TOP command by pressing the <PF1> key followed by the <5> key. This sequence is denoted by <PF1/5>. Note that this does not change the current direction; it simply moves the cursor to the first character in the buffer. Press <PF1/4> to execute the BOTTOM command.

There are three other simple cursor movement commands: END OF LINE, SECTION, and PAGE. These are not used as frequently as the WORD, LINE, TOP, and BOTTOM commands, but, like the WORD and LINE commands, they too depend upon the current direction. Press <2> for the END OF LINE command. It advances the cursor to the end of the current line (or the end of the preceding line, if the current direction is BACKUP). The SECTION command moves the cursor 16 lines at once; a good way to move through a large buffer. The PAGE command moves the cursor to the beginning of the next page, as defined by the *page delimiters*. The page delimiter is created with <CTRL/L>, appears as <FF> on the screen, and causes a printer to advance to the next page before continuing to print. (See Appendix D.)

The Delete Commands

Now that you know how to move forward and backward through the buffer by characters, words, lines, sections, and pages, the next step is to learn how to delete these units. The ease of such deletions is one of the nicest features of the Keypad Mode.

The <DEL> key has the same effect in all modes: it erases the character just to the left of the cursor. The DELETE CHARACTER command, executed by pressing the comma key on the keypad, is almost the same. It deletes the current character, which is the character that the cursor is on. Similarly, the DELETE WORD command, executed by pressing <->, erases all of the characters from the current character to the first character of the next word on the right (regardless of the current direction). This moves the rest of the line to the right without moving the cursor, positioning that first letter of the next word under the current cursor position. Move the cursor to the "t" in "truly" in the first sentence of your letter, and then use the DELETE WORD command to delete the word "truly." The DELETE LINE, executed by pressing <PF4>, works the same way.

The three DELETE commands are a little more forgiving than the <DEL> key; they allow you to "undelete" your last deletion. Try this by moving the cursor to the "w" in "would" in the first sentence; then press <PF1/-> (i.e., press the <PF1> key and then the <-> key). This magic is achieved with the *word buffer*, which is a special buffer used to hold the last word deleted. In a similar manner, the UNDELETE CHARACTER command <PF1/,> and the UNDELETE LINE <PF1/PF4> command use the character buffer and the line buffer, respectively, to restore deleted characters and lines.

The DELETE TO END OF LINE command, executed by the <PF1/2> key, is almost the same as the DELETE LINE command. The only difference is

that it affects only the line deleted; it does not move the beginning of the next line up to the current cursor position. It uses the same line buffer that the DELETE LINE command uses. Move the cursor to the middle of any line in your letter, and press <PF1/2>. Then, restore the deleted text with <PF1/PF4>.

Because each of these delete commands allows the deleted text to be restored later wherever the cursor is positioned, they really should be regarded as MOVE commands. For example, the word "truly" was moved to a new position using the DELETE WORD and UNDELETE WORD commands.

Cutting and Pasting

Another, more surgical way to move or delete parts of your text is to use the SELECT, CUT, and PASTE commands. To illustrate these, suppose that you want to delete that part of the first sentence of your letter which begins with the space after the word "computers," and ends with the "e" just before the period, thereby removing the clause beginning with "since." First, move the cursor to the space just before the word "since." Press <.> to execute the SELECT command. Then, using the WORD command, <1>, and the left-arrow cursor control key, advance the cursor to the period at the end of the sentence. Notice that the SELECT command highlights the text selected. This block of text is called the *select range.* Press <6> to execute the CUT command. This moves the selected block of text into the paste buffer. At this point, the select range is no longer considered to be active.

As with any other buffer, the contents of the paste buffer can easily be restored. Simply execute the PASTE command <PF1/6>. This process is called "cut and paste" because it simulates the editorial process of cutting out a block of text and then pasting it somewhere else. It is not quite the same, however, because the select range is still in the paste buffer; only a copy was "pasted" back into the main buffer. To verify this, press <PF1/4> to move to the BOTTOM of the buffer and then execute the PASTE command again (Fig. 4-6). Notice that the space before the word "since" and the invisible end-of-line

```
                                                    July 22, 1986

Dear Ada,

     You truly would be amazed at the progress made with computers since you
worked with Charles on his Analytical Engine.  Your predictions about their
usefulness have all come true!

     You were absolutely right about how well computers are able to process
textual information.  Indeed, I am using a computer now to write this letter to
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

 since you
worked with Charles on his Analytical Engine
[EOB]
```

Fig. 4-6. Executing the SELECT command. This highlights the "select range". On most word processors, this is called the "block".

marker after the word "you" (forcing a new line) are part of the text in the paste buffer.

A more efficient way to examine a buffer is to use the Line Mode TYPE command. Press <CTRL/Z> to exit the Keypad Mode and return to the EDT Line Mode, then give the TYPE command exactly as shown in Fig. 4-7. This causes the entire paste buffer to be displayed. The /STAY qualifier is necessary to keep the main buffer current; without it, the paste buffer would become the current buffer.

```
*TYPE =PASTE/STAY
    0.1       since
    0.2     worked with Charles on his Analytical Engine
[EOB]
```

Fig. 4-7. Examining the PASTE buffer. The equal sign (=) signifies a buffer name.

Return to the Keypad Mode with the CHANGE command. Then, use the DELETE LINE command twice to remove the "garbage" at the bottom of the file.

It is not necessary to exit the Keypad Mode to execute a Line Mode command. The COMMAND command allows you to do this while remaining in Keypad Mode. Press <PF1/7> to execute this command. Notice the "Command:" prompt on the bottom line. Type the same Line Mode command as before (see Fig. 4-8). Then, press <ENTER> (the large key in the lower right corner of the keypad). You get the same information as before, without disturbing your full-screen editing. Press <RET> to restore your environment.

Reformatting

In the previous section, when you "cut" your select range into the paste buffer, part of the next sentence was left running off the screen. To see how to remedy that common situation, repeat the first steps of that section again, so that your screen looks like Fig. 4-9. Recall that the little diamond symbol in column 80 indicates that you have more text past that point on the line. You could reformat the paragraph directly, using the <RET> and DELETE commands, but it is much easier to let the VAX do all that work for you.

With the cursor at the end of the first sentence, press <.> to start selecting the range again. Then, press the down-arrow cursor control key once, thereby selecting the remainder of the first paragraph. This is the part of the text which needs to be reformatted. Now, press <PF1/8> to execute the FILL command. Voila! The text in the select range is immediately modified to "fill" each 80-column line without running over the right margin (Fig. 4-10).

The FILL command is especially efficient when large documents need to be reformatted. It is just as easy to select an entire 200-line buffer as the range as it was to select half of a paragraph.

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with computers since you
worked with Charles on his Analytical Engine.  Your predictions about their
usefulness have all come true!

     You were absolutely right about how well computers are able to process
textual information.  Indeed, I am using a computer now to write this letter to
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]



Command: TYPE =PASTE/STAY
    0.1       since
    0.2     worked with Charles on his Analytical Engine
[EOB]
Press return to continue
```

Fig. 4-8. Using the COMMAND command in the EDT Keypad Mode to execute an EDT Line Mode command.

The FIND Commands

In Chapter 2, you learned how to search for a given word or character string in the Line Editing Mode. In the Keypad Editing Mode, this function is performed by the FIND and FIND NEXT commands.

Suppose you want to search for the character string "UL" in your buffer. First, execute the TOP command <PF1/5> to move the cursor to the beginning of the buffer. Next, execute the FIND command by pressing the <PF1/PF3> key. This will cause the editor to prompt you for the character string that you want to find. Type UL, as shown in Fig. 4-11, and then press the <ENTER>

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with computers. Your predi♦
usefulness have all come true!

     You were absolutely right about how well computers are able to process
textual information.  Indeed, I am using a computer now to write this letter to
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]
```

Fig. 4-9. A file with a line longer than 80 characters. The diamond character appears in column 80 (the right-most column).

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with computers.  Your
predictions about their usefulness have all come true!

     You were absolutely right about how well computers are able to process
textual information.  Indeed, I am using a computer now to write this letter to
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]
```

Fig. 4-10. Executing the SELECT and FILL commands in the EDT Keypad Mode to reformat the text. This powerful feature is not available on most word processors.

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with computers.  Your
predictions about their usefulness have all come true!

     You were absolutely right about how well computers are able to process
textual information.  Indeed, I am using a computer now to write this letter to
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]



Search for: UL
```

Fig. 4-11. Executing the FIND and FIND NEXT commands in the EDT Keypad Mode. This search technique is similar to searching in Line Mode, but it allows you to see more of file as it is being searched.

key. As soon as you press <ENTER> (not <RET>), two things happen. First, the string "UL" is put into a special *search buffer.* Next, the cursor is immediately moved (in the current direction) to the next occurrence of the given string. Notice that the FIND command does not distinguish between uppercase and lowercase letters. Press <PF3> to execute the FIND NEXT command. Each time this is done, the cursor will move to the next occurrence of the string. Press <PF3> a few more times to see how this works.

The SUBSTITUTE command is often used with the FIND commands. It is analogous to the Line Mode's SUBSTITUTE command (see Step 11 in Chapter 2). To see how it is used, suppose you wish to change the word "computer" to "digital computer" everywhere it occurs in your letter to Ada.

First, you insert "digital computer" into the paste buffer by pressing <.> for the SELECT command, typing "digital computer", and pressing <6>. It doesn't matter where the cursor was in the buffer; everything typed between <.> and <6> will be highlighted and removed to the paste buffer. After pressing <PF1/5> to move to the TOP of the buffer, press <PF1/PF3> to execute the FIND command (as you did in the previous example). Respond to the "Search for:" prompt with the word "computer," and then <ENTER>. This puts "computer" in the search buffer. Now you have "digital computer" in the paste buffer and "computer" in the search buffer. Each time you press <PF1/ENTER> for the SUBSTITUTE command, the cursor will move to the "c" of the next occurrence of "computer" and automatically replace it with the string "digital computer" from the paste buffer. Do that now: press <PF1/ENTER> several times.

Your screen should now look like Fig. 4-12. The message "String was not found" appears on the bottom line to indicate that you have found all the occurrences of "computer" (in the current direction). Once again, however, you have some lines with more than 80 characters (indicated by the little diamond symbol in column 80). Use the SELECT and FILL commands to reformat the letter: press <PF1/5/./PF1/4/PF1/8>. The result should look like Fig. 4-13.

SELF-TEST EXERCISES

1. Add 20 or so more lines to your file; write anything, even blank lines. Then, investigate the SECTION command <8> together with the BACKUP <5>, ADVANCE <4>, TOP <PF1/5>, and BOTTOM <PF1/4> commands.

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with digital computers.  Yo#
predictions about their usefulness have all come true!

     You were absolutely right about how well digital computers are able to pro#
textual information.  Indeed, I am using a digital computer now to write this l#
you.  It was made by some very clever people here in Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]



String was not found
```

Fig. 4-12. Executing the SUBSTITUTE command in the EDT Keypad Mode. This is similar to the SUBSTITUTE command in Line Mode.

```
                                                    July 22, 1986

Dear Ada,

     You would truly be amazed at the progress made with digital computers.
Your predictions about their usefulness have all come true!

     You were absolutely right about how well digital computers are able to
process textual information.  Indeed, I am using a digital computer now to
write this letter to you.  It was made by some very clever people here in
Massachusetts.

                                                    Your humble servant,

                                                    Herman

[EOB]


String was not found
```

Fig. 4-13. Executing the SELECT and FILL commands in the EDT Keypad Mode to reformat the text.

2. Insert form feeds by pressing <CTRL/L> at various locations in your file. Then, investigate the PAGE command <7> together with the BACKUP <5>, ADVANCE <4>, TOP <PF1/5>, and BOTTOM <PF1/4> commands.
3. Use the APPEND command <9> with the SELECT, CUT, and PASTE commands. First, insert some text in the paste buffer as you did in the section on cutting and pasting. Next, use SELECT to select another range of text, but don't use CUT on that range. Instead, press <9> to APPEND the second range to the text already in the paste buffer. Use the COMMAND command together with the Line Mode command T=PASTE/ST at the "Command:" prompt to see what is in the paste buffer. You can repeat these steps to append as many ranges of text as you want, and you can paste these just as before.
4. In Self-Test Exercise 7 of Chapter 3, you created a file named HEADER.TXT to be used as a common header in files for programs. You used the Line Mode command INCLUDE to insert this text into other files. You can also do this from within the Keypad Mode using the COMMAND command. Move the cursor to the position in the buffer where you would like the external file to be inserted. Then, press <PF1/7> and type INC HEADER.TXT at the "Command:" prompt. Remember to terminate the command with <ENTER>, not <RET>.
5. Repeat the last example in this chapter replacing the first, third, fourth, and sixth occurrence of the string "you" with the string "y'all." Do this by executing SUBSTITUTE, FIND NEXT, SUBSTITUTE, SUBSTITUTE, FIND NEXT, and SUBSTITUTE. In other words, use FIND NEXT instead of SUBSTITUTE whenever you do not want to make the substitution. This demonstrates that the Keypad Mode SUBSTITUTE command is more flexible than the Line Mode SUBSTITUTE command; you can easily decide which occurrences you want to change as they are found.

6. Another useful application of the SELECT command is to change the case of letters (uppercase to lowercase, or vice versa) in a selected range. This is achieved with the CHANGE CASE command, executed by <PF1/1>. First, set up a select <.> range. Then, just press <PF1/1>. Notice that all uppercase letters are changed to lowercase, and all lowercase letters are changed to uppercase.
7. The "automatic repeat" feature of the keyboard and cursor control keys is simulated for the other Keypad Editing Mode keys by means of the <PF1> key. The only difference is that you must decide before you use it how many times you want the command to be repeated. To see how this works, move the cursor to the beginning of the second sentence and press <PF1>5<-> to delete the next five words. Be sure to press the "5" key on the standard keyboard (above the "R" key), not the keyboard key <5>. Notice that the repeat number is displayed on the bottom line of the screen.
8. Experiment with the auto-repeat function on other commands; for example, the other DELETE commands, the cursor control keys, and the paste command. For auto-repeat on the PASTE command, insert "****!" into the Paste Buffer and then press <PF1>7<PF1/6>.
9. Use the HELP command to learn how all the other full-screen editing commands work. For example, investigate the SPECIAL INSERTION command by pressing <PF2/3>, and the DEFINE KEY command by pressing <PF2/CTRL/K>. Check Appendix D for details.

Chapter 5

Running a Pascal Program

Mathematicians are like Frenchmen: whatever you say to them they translate into their own language, and forthwith it is something entirely different.

Goethe

In this chapter you will learn how to write, compile, link, and run simple computer programs written in the Pascal programming language. If you have run programs in BASIC on a microcomputer, you will see that the process of running Pascal programs on the VAX is similar; only a few more steps are needed. If you have run programs (in any language) on some other large computer, you will appreciate how easy it is to use the VAX.

THE PASCAL PROGRAMMING LANGUAGE

The Pascal programming language was invented by Niklaus Wirth in 1968. He named it after the French mathematician Blaise Pascal (1623 – 1662), who is believed to have been the first person to design and build a computer. (Actually, it was only a primitive calculator.) Wirth designed the Pascal language as a pedagogical tool, to help university students learn how to program correctly. However, during the past decade, it has become one of the most popular programming languages throughout the computer industry.

As a first example, you will run a program which does only one simple thing:

it displays the message "Greetings from the VAX" on the screen. The complete program is given in Fig. 5-1.

```
program Message(output);
  begin
    write('Greetings from the VAX')
  end.
```

Fig. 5-1. A simple Pascal program. Note that Pascal programs do not have line numbers.

If you have used programs written in the BASIC programming language, you immediately will notice two big differences between Pascal and BASIC: Pascal uses mostly lowercase letters, and Pascal has no line numbers. The first difference is trivial. You may use uppercase or lowercase letters anywhere you want; the VAX compiler doesn't care. The second difference is very significant. Pascal is a free-form language. You may put several statements on one line; you may also extend one statement over several lines. In general, blanks and line separators are ignored by the compiler.

Every Pascal program has two main parts: its *program heading* and its *main block*. The program heading is usually just the first line. It has three parts: the reserved word "program," the name of the program, and the list of file-identifiers. In the program in Fig. 5-1, the program name is Message, and the list of file-identifiers is "(output)," which means that the program will have output.

Everything after the program heading constitutes the main block. In this simple example, the main block contains the reserved words "begin" and "end," and the single executable statement:

```
write('Greetings from the VAX')
```

When the program runs, this statement causes the message "Greetings from the VAX" to be displayed on the screen. The standard identifier "write" tells the VAX to "write the following:". If the thing to be written is a message (instead of the value of a variable), then it must be enclosed by apostrophes. Everything between the two apostrophes will be displayed.

A SAMPLE PASCAL PROGRAM

In the previous chapters, you learned how to use the EDT editor to create and modify files. Now you shall learn how to write and run programs like the one above.

Writing the Program

Log in to the VAX, as described in Chapter 1, then enter the EDT Line Mode, as described in Chapter 2. Name your new file PROGRAM1.PAS. The

filetype PAS stands for "Pascal." All Pascal programs should have the filetype PAS. Of course, any filename of up to 39 alphanumeric characters is acceptable.

Enter the EDT Character Keypad Mode, as described in Chapter 4, and type in the program exactly as it appears in Fig. 5-1. Don't forget the period at the end. Then, exit from the Keypad Mode, and then from the Line Mode.

You now have a Pascal program in the file PROGRAM1.PAS. This is called the **source code** of the program. The next step is to *compile* your program. This means to have it translated into the *machine language* which the computer can "understand."

Compiling the Program

To compile a Pascal program, the Pascal *compiler* must be invoked. This is a separate program, available to all users of the VAX, which translates the Pascal source code into machine language. The DCL command which causes this to happen is the PASCAL command.

The PASCAL command is analogous to the EDIT command. They both tell the VAX/VMS operating system to run a large resident applications program which will produce a new file. The new file which the Pascal compiler creates will allow the VAX to run the program. The filename of the created file will be the same as that of the Pascal program: PROGRAM1 in this first example. The file type is automatically set to be OBJ. This stands for object module. On the VAX, an *object module* is a file that contains the translation of source code into machine language.

Now, compile your program by invoking the Pascal compiler on your source code, as shown in Fig. 5-2. Note that only the filename, PROGRAM1, is given as a parameter to the PASCAL command. The system assumes that your file type is PAS, because you are using the Pascal compiler.

```
$ PASCAL PROGRAM1
```

Fig. 5-2. Executing the PASCAL command to compile the program in the PROGRAM1.PAS file. This translates the text file into machine language. This new file is given the filetype OBJ for "OBJect module."

After executing the PASCAL command as shown above, one of two things will happen. Either your program will compile successfully, signaled by the reassuring dollar sign prompt, or the compiler will find some *compile-time errors*. The most common type of compile-time error is a *syntax error*, i.e., a violation of the rules of the programming language. (A common syntax error made by beginners is to forget to include the period at the end of the program.) If you copied the example correctly, however, your program did compile and you should be ready for the next step. If not, go back to the previous section and try again.

Linking the Program

The Pascal compiler translates source code directly into machine language, but that is not good enough to allow it to run yet. The VAX requires that "all external references be resolved." This means that the object module must be *linked* to other machine language instructions which allow the system to carry out the steps not completely explained by the module itself. For example, this simple program calls a predefined procedure named "write." This, and other such external procedures and functions, can be invoked only if they are first linked to the machine language code which tells the system how to do it. The linking is performed by the VAX/VMS Linker, another large applications program available to all VAX users.

Execute the LINK command as shown in Fig. 5-3. Note that, again, only the filename PROGRAM1 is passed as a parameter.

```
$ LINK PROGRAM1
```

Fig. 5-3. Executing the LINK command to link the program. This combines the translated program (the object module) with other modules necessary for execution. The resulting combined file (also in machine code) is given the filetype EXE for "EXEcutable image".

Like the PASCAL command, the LINK command should execute without any response. No news is good news. Also like the Pascal compiler, the VAX/VMS Linker creates yet another file. This file, too, will have the same filename, but its filetype will be EXE for "EXEcutable image." The *executable image* of your program is just what it sounds like: a modification of your original source code which finally can be run.

At this point, you should check your directory. It should now contain the two new files PROGRAM1.OBJ and PROGRAM1.EXE. Both the object module, PROGRAM1.OBJ, and the executable image, PROGRAM1.EXE, are in machine language. Only the computer can understand these strings of zeros and ones. To see this, try to TYPE these files. As you can see in Fig. 5-4 and Fig. 5-5, when the terminal tries to translate the machine language in these files into ASCII characters, the result is gibberish. Only humans can understand Pascal source code, and only computers can understand machine language.

```
$ TYPE PROGRAM1.OBJ
MESSAGE0123-Jul-1986 20:1723-Jul-1986 20:17VAX Pascal V3.3-
92q<MESSAGEP]Greeting
PAS$FV_OUTPUPAS$WRITE_STRINGi3$CODE}9


                                        9  ~k8$CODE3>#MESSAGE?#
```

Fig. 5-4. The effect of attempting to display the contents of the object module PROGRAM1.OBJ. The only files that can be TYPEd (read by humans) are text (ASCII) files.

```
$ TYPE PROGRAM1.EXE

(ODX0205(#OR%h_~PROGRAM101`W#OR%04-00




1?
  }+e
     PASRTL_001v!



LIBRTL_001

@PASRTLLIBRTL<MESSAGE9.o`]${PP@@T

                              9 ~k8$CODE3>#MESSAGE?
```

Fig. 5-5. The effect of attempting to display the contents of the executable image PROGRAM1.EXE. The terminal is attempting to translate machine code into ASCII characters.

Running the Program

Now that you have the executable image PROGRAM1.EXE in your directory, you can finally run your program. Execute the RUN command, as shown in Fig. 5-6.

```
$ RUN PROGRAM1
Greetings from the VAX.
```

Fig. 5-6. Running the program. It's almost instantaneous.

You have seen that the process of writing, compiling, linking, and finally running a Pascal program requires several steps and involves three different files. The relationships between these steps and files is shown in Fig. 5-7.

Actually, each of the four DCL commands, EDIT, PASCAL, LINK, and RUN, produces output. The output from each of the first three commands is put in a separate file, and all three of these files have the same filename. The output from the RUN command goes to the standard output file. In most situations, this means that the output from running the program is displayed on the terminal

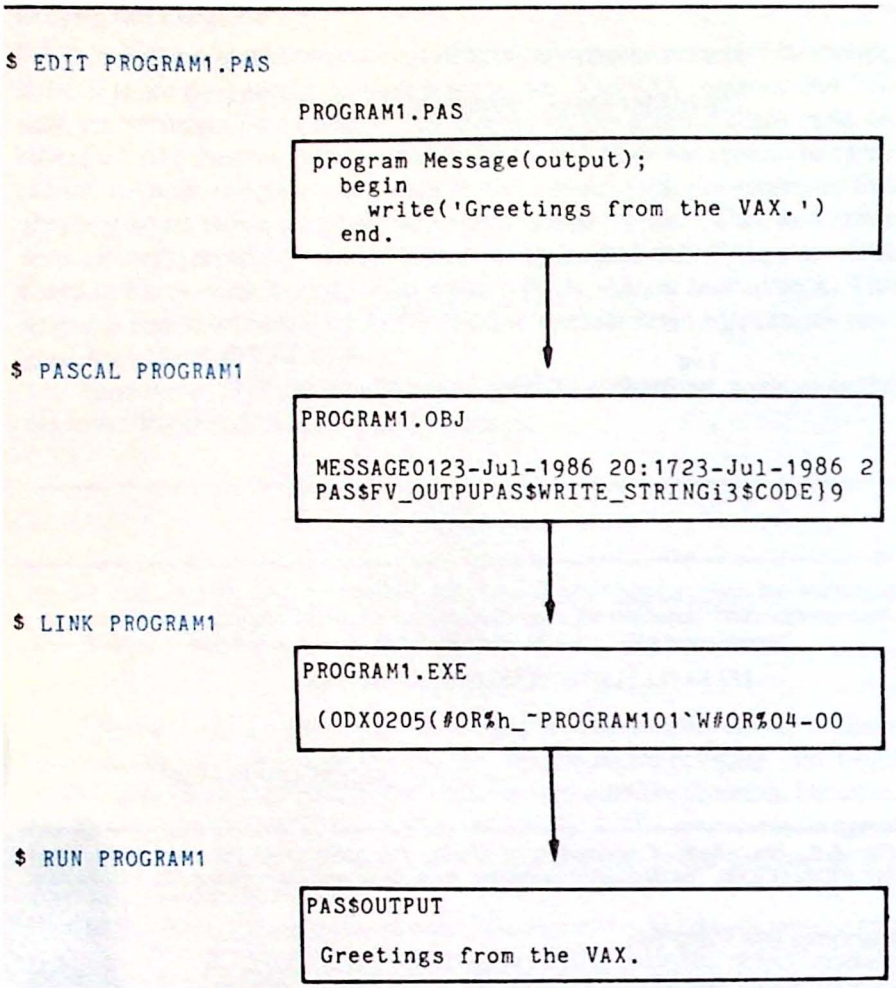

Fig. 5-7. The progression of steps required to create, compile, link, and run a Pascal program. The process creates three new files in your account: the source code, the object module, and the executable image.

screen, which is the standard output file. However, it is quite easy on the VAX to divert the output to another file, which can be used later. Meanwhile, the *logical name* for the standard output file is PAS$OUTPUT when the output comes from a Pascal program.

After your program has been successfully compiled and linked, you can run it as often as you wish. The PASCAL and LINK commands are necessary only if you change your source code. Consequently, you could DELETE the PROGRAM.PAS and PROGRAM.OBJ files, and still run your program as much as you want. Deleting the object module is a common practice, but it is probably a mistake to delete your source code. It is the only readable version, so without it, you might find it difficult to tell how the program works!

If writing, compiling, linking, and running your program all went well, consider yourself lucky. More often than not, compile-time errors occur. These require you to go back into the editor, fix your source code, and then re-compile your program. The moral is: Know the syntax of your programming language, and take time to write your program carefully. Each minute spent during the programming stage of your project can eliminate at least five minutes at the terminal.

AN INTERACTIVE PROGRAM

One of the nice features of the VAX/VMS system is that it is *interactive.* This means that the user can interact with the system while a program is running, giving it input and reading its output. The program shown in Fig. 5-8 is a simple example of an interactive program. Its purpose is to simulate a simple calculator by dividing two numbers. This means that it must be able to read these two numbers, and then display their quotient. So, unlike our first example, this program has input as well as output.

```
program Interact(input,output);
  var
    X,Y: integer;
  begin
    repeat
      write('Enter an integer: ')
      readln(X);
      write('Enter another integer: ');
      readln(Y);
      writln('The ratio is X/Y);
      writeln
    until (0 = 1)
  end
```

Fig. 5-8. An interactive Pascal program. The dialog is programmed in advance.

Program Variables

Any program that performs calculations must use variables to store the values being calculated. A *variable* is simply a storage location in the main memory of the computer. The program uses symbolic names (X and Y in this example) to identify the variables. Such a name is called an *identifier.* The Pascal compiler knows what the reserved words, like "program" and "begin," mean. It also has a list of standard identifiers, like "output" and "write," whose meaning it knows. All other identifiers, which are not known, must be declared in the program; that is, the programmer must tell the compiler what the identifiers stand for. This is done in the *declaration section* of the program, which begins with the reserved word *var* (for "variable"). The declaration in this program tells the compiler that x and y are integer variables.

This program has several features which make it more complicated than the first example. Besides the use of variables, it also uses a *repeat loop*. In a repeat loop, the statements inside the loop are repeated until the condition next to the reserved word "until" becomes true. Of course, in this case, the condition "0 = 1" will never be true. This creates an infinite loop which will never terminate normally; that is, the program will never reach its end. This logical error is intentional, to study the behavior of the system.

Another thing which makes this program more complicated than the first example is its use of the *readln* and *writeln* procedures (pronounced "read-line" and "write-line"). The statement "readln(X)" will copy the integer typed at the keyboard into the storage location identified by the symbol X. The writeln procedure works like the write procedure, but it always advances the cursor to the beginning of the next line after writing the quantity in the parentheses. In the last instance, where it is used without parentheses, it simply skips a line on the screen (or the printed copy).

Enter the EDT editor and type this program into a new file called PROGRAM2.PAS. Copy this program exactly as you see it here, even if you think there might be some errors. As you might have suspected, this program does, in fact, have some errors. They will be used to illustrate different aspects of the process of compiling, linking, running, and debugging.

Try to compile this program, as shown in Fig. 5-9.

```
$ PASCAL PROGRAM2
00007      0  0       readln(X);
                      1
%PASCAL-E-SYNSEMI, (1) Syntax: ";" expected
00010      0  2       writln('The ratio is X/Y);
                      1                        2
%PASCAL-E-UNDECLID, (1) Undeclared identifier WRITLN
%PASCAL-E-QUOBEFEOL, (2) Quoted string not terminated before end
of line
00012      0  2     until (0 = 1)
                    1
%PASCAL-E-SYNCOMRP, (1) Syntax: "," or ")" expected
00013      0  0   end
                     1
%PASCAL-W-SYNPERIOD, (1) Syntax: "." expected
%PASCAL-E-ENDDIAGS, PASCAL completed with 5 diagnostics
```

Fig. 5-9. The system's response to various compile-time errors. The string %PASCAL indicates that the error messages are coming from the Pascal compiler.

Compile-Time Errors

The Pascal compiler has uncovered five compile-time errors, two on line 10, and one each on lines 7, 12, and 13. (These are the line numbers assigned by the EDT editor; the program itself has no line numbers.) Each error is pointed to by a digit directly below where the error was discovered. For example, the second error discovered on line 10 was detected directly after the semicolon at the end of the line, so the digit "2" points to that position.

Each error causes an error message to be printed. Like all VAX/VMS error messages, these messages have several parts. First, the percent sign signals that this is an error message. Next, the name of the system which found the error is given, in this case PASCAL, denoting the Pascal compiler. Then, after the hyphen, comes one of the letters S, I, W, E, or F to indicate the *severity level* of the error: either Success, Information, Warning, Error, or Fatal. An abbreviation of the error message is given next, followed by a digit in parentheses corresponding to the digit on the preceding line which points to the place where the error was detected. Finally, the actual error message is displayed.

The first error detected is a SYNSEMI error; that is, the SYNtax error of a missing SEMIcolon. This is one of the most common compile-time errors in Pascal programs. The Pascal language uses semicolons instead of line numbers to separate statements. These are usually put at the end of each line. In this program, the required semicolon was left out at the end of line 6. The compiler couldn't tell that it was missing, however, until it found something else instead. As soon as it found the "r" at the beginning of the next line, it knew that the semicolon was missing. So, the error message points to that "r" on line 7.

The second error is UNDECLID for UNDECLared IDentifier. This is also a common compile-time error, usually resulting from a typing error in the source code. In this case, writeln was misspelled as writln. The compiler determined that this must be an identifier, because it is not one of the Pascal reserved words. It is not one of the predeclared identifiers, so the compiler decided that it must be declared. It is not listed in the declaration of the program, however; hence, the error.

The third error, QUOBEFEOL, is on the same line. The apostrophe on the right of the string to be printed was omitted. This is another common mistake.

The fourth error statement (SYNCOMRP, for SYNtax: COMma oR Parenthesis expected) is more difficult to understand. The problem here is that the previous mistake was so bad that it confused the compiler into erroneously detecting another mistake. This is understandable, because the compiler can't really think. False error detection like this occurs frequently when there are a lot of errors in the source code. The best solution is to try to repair only the first few errors detected, and then recompile the program. The fifth error, SYNPERIOD, is obvious. The period, which must be the last thing in the program, was omitted.

The Pascal compiler has told you what and where the errors are, so it is a simple matter to correct them. Go into the EDT Line Mode and use the SUBSTITUTE command to correct them, as shown in Fig. 5-10. Then, recompile your program. (PAS is the abbreviation for the PASCAL command.) This time, it should compile. This shows that the fourth "error" was not an error at all; the compiler was just confused from the third error.

Now, link and run your program, as shown in Fig. 5-11.

Logical Errors

The program is running, but something is still wrong: it's printing the symbols X/Y instead of the actual numerical ratio. This is a *logical error.* It is a flaw in the logical plan of the program itself.

```
$ ED PROGRAM2.PAS
    1       program Interact(input,output);
*6
    6               write('Enter an integer: ')
*S/)/);/
    6               write('Enter an integer: ');
1 substitution
*S/tln/teln/10
   10               writeln('The ratio is X/Y);
1 substitution
*S/)/')/
   10               writeln('The ratio is X/Y');
1 substitution
*S/d/d./13
   13          end.
1 substitution
*EXIT
SYS$USER:[REAGANRON]PROGRAM2.PAS;2 13 lines

$ PAS PROGRAM2
```

Fig. 5-10. Debugging the program, correcting syntax errors. A common task.

```
$ LIN PROGRAM2
$ R PROGRAM2
Enter an integer: 22
Enter another integer: 7
The ratio is X/Y

Enter an integer:
```

Fig. 5-11. Running the program. As far as the VAX is concerned, the program is running perfectly. It cannot detect logical errors.

The VAX can spot compile-time errors and tell you about them, but it has no way of detecting logical errors. As far as the VAX is concerned, your program is running perfectly. It is doing just what you told it to do, display the string, "The ratio is X/Y."

Logical errors are usually more difficult to fix than compile-time errors. The VAX has no way of detecting them, so it is up to human users to find them. This particular logical error, however, is relatively obvious. You want the VAX to actually compute the value of the quotient X/Y, and then print the numerical result. This is quite different from printing a character string like X/Y. In Pascal, actual numerical values of expressions like X/Y must be listed separately from other quantities in the parameter list of the writeln procedure. The parameter list is indicated by the parentheses, and each individual parameter in that list must be separated by commas. Moreover, because you want the value of the expression X/Y, not the character string X/Y, it must be listed without apostrophes.

Like any other kind of error, logical errors must be repaired in the editor. The program is still running, however; it is waiting for another integer to be

typed. To interrupt the program, press <CTRL/Y>. Then, go back into EDT Line Mode, and fix the error as shown in Fig. 5-12. Compile and link your program again. Then, run it with some sample numbers, as in Fig. 5-13.

```
Enter an integer: <CTRL/Y>
 Interrupt

$ ED PROGRAM2.PAS
    1       program Interact(input,output);
*'X/Y'
   10               writeln('The ratio is X/Y');
* S.X/Y'.',X/Y.
   10               writeln('The ratio is ',X/Y);
1 substitution
*EX
SYS$USER [REAGANRON]PROGRAM2.PAS;3 13 lines
```

Fig. 5-12. Debugging the program, correcting a logical error. Now the string X/Y will be interpreted as an expression to be evaluated instead of just a character string to be printed. Note the use of the period as a delimiter in the S command.

```
$ PAS PROGRAM2
$ LIN PROGRAM2
$ R PROGRAM2
Enter an integer: 22
Enter another integer: 7
The ratio is  3.14286E+00

Enter an integer: 1
Enter another integer: 128
The ratio is  7.81250E-03

Enter an integer: -65536
Enter another integer: 1
The ratio is -6.55360E+04

Enter an integer: <CTRL/Y>
 Interrupt
```

Fig. 5-13. Running the program.

Notice that the output is in *floating-point notation*: 7.81250E−03 means "7.81250 times 10 to the power −3" (the E is for "Exponent"). In *fixed-point notation*, this would be 0.0078125. The number after the E simply tells how many places to move the decimal point to the left (for a negative sign, as shown here) or right (for a positive sign).

This program still has room for improvement. Besides printing the result in the awkward floating-point notation, the program doesn't terminate normally. These problems will be considered further in the Self-Test Exercises at the end of the chapter. In addition to these problems, this program still has another, more serious flaw. Run it again, as in Fig. 5-14.

```
$ R PROGRAM2
Enter an integer: 8
Enter another integer: 0
%SYSTEM-F-FLTDIV_F, arithmetic fault, floating divide by zero at PC=000002C1 PS
L=03C00002
%TRACE-F-TRACEBACK, symbolic stack dump follows
module name     routine name                    line       rel PC     abs PC

The ratio is    INTERACT                          10     00000085   000002C1
```

Fig. 5-14. The system's response to a run-time error. The VAX can't divide by zero.

Run-Time Errors

Everyone knows you can't divide by zero. When a program tries to get the VAX to do so, it reacts this way. This is called a *run-time error.* It is also called a *crash.* It is the most severe type of error, occurring inside the central processing unit of the VAX. The VAX responds by providing a *symbolic stack dump*: an accounting of the vital information which an experienced programmer could use to glean more information about the state of things when the crash occurred.

Run-time errors usually result from erroneous input. Compile-time errors are caused by mistakes made during the writing of the program in the editor; syntax errors are an example. Logical errors are made in the planning stages, when the program is being designed.

SELF-TEST EXERCISES

1. Write another simple program, like the one shown in Fig. 5-1, which prints your name and address. Compile, link, and run it.
2. Write another interactive program, like the one given shown in Fig. 5-8, which adds and subtracts two integers. Compile, link, and run it.
3. Rewrite the program in Fig. 5-8 so that it will print the quotient in fixed-point notation. This is done in Pascal by appending a formatting string like ":8:4" to the expression to be evaluated. For example, writeln(X/Y:8:4) would print the value of X/Y using a total of eight columns and allowing four digits to the right of the decimal point. Do this with several different formats.
4. Rewrite the program in Fig. 5-8 so that the loop terminates naturally. Do this

```
program Exercise5(output);
  var
    Speed,Duration,Distance: integer
  begin
    write('Enter speed: );
    readln(Speed);
    write('Enter time: ');
    readln(Time);
    Distance := Speed*Time;
    writeln('The distance travelled was ',Distance
  end
```

Fig. 5-15. A program to be debugged. What's wrong with this picture?

by changing the loop control condition to (X = 0). Also, add a statement at the beginning to tell the user that this will stop the loop.

5. Correct the program in Fig. 5-8 to make it "crashproof." Use the conditional statement *if (Y<>0) then.*
6. Use the Pascal compiler to debug the program shown in Fig. 5-15.

Chapter 6

Introduction to Minitab

Statistical thinking will one day be as necessary for efficient citizenship as the ability to read and write.

H. G. Wells

Minitab is a statistical software system written under the direction of T. A. Ryan, Jr. at the Pennsylvania State University. It was developed in the 1970s to assist students in elementary statistics courses at Penn State. Since then, it has been widely adopted elsewhere because of its simplicity, flexibility, and power.

This chapter offers an introduction to the Minitab system. Appendix E lists most of its common commands. For further information about the system, see the references in the Bibliography.

THE MINITAB MODE

On the VAX, Minitab can be regarded as a separate mode, internal to the DCL Mode, and parallel to the EDT Line Mode. This is shown in Fig. 6-1. The DCL command which invokes the Minitab system is simply MINITAB. Type that command now to enter the Minitab Mode (Fig. 6-2). Note that the prompt for the Minitab Mode is **MTB>**. Give the STOP command to exit from the Minitab Mode and return to the DCL Mode.

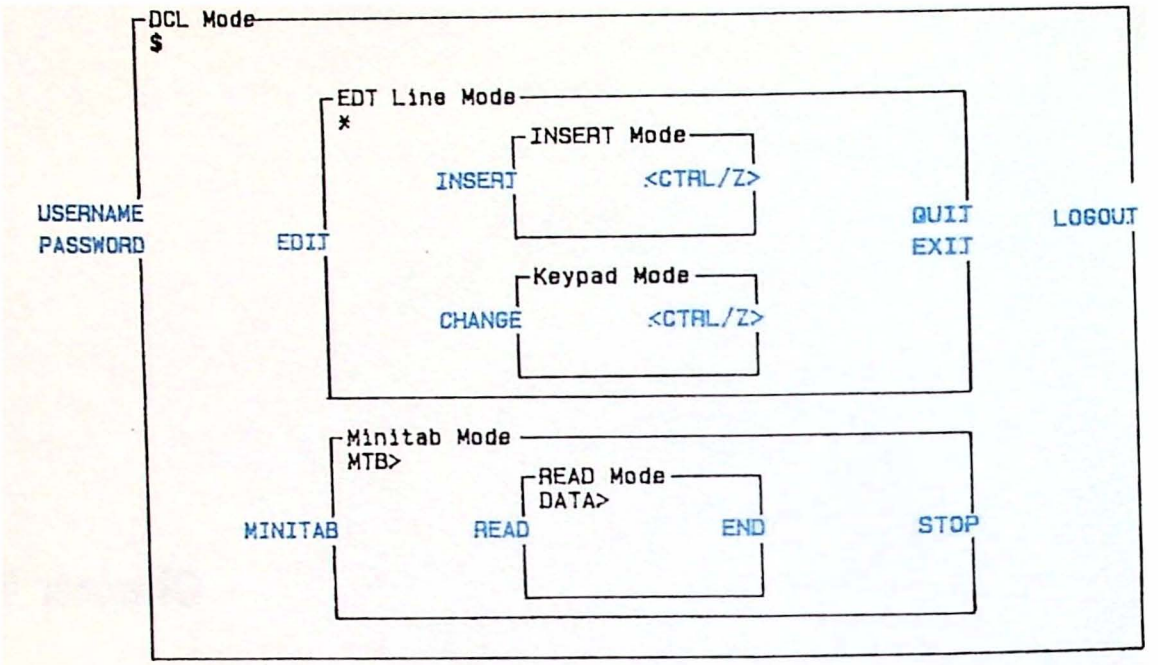

Fig. 6-1. The DCL Mode, the EDT Line Mode, the INSERT Mode, the Keypad Mode, and the Minitab Mode.

```
$ MINITAB
MINITAB RELEASE 5.1 *** COPYRIGHT - MINITAB, INC. 1985
U.S. FEDERAL GOVERNMENT USERS SEE HELP FGU
JULY 27, 1986 *** University of Richmond
STORAGE AVAILABLE 896430

 Type NEWS for information on new features in Minitab Release #5.1
MTB > STOP
*** Minitab Release 5.1 *** Minitab, Inc. ***
Storage available 896430
```

Fig. 6-2. Executing the MINITAB command.

The Purpose of Minitab

The purpose of Minitab is to perform statistical analysis on numerical data. To use Minitab then, the first thing you will need is some numerical data. Use the sample data given in Fig. 6-3.

Student No.	SAT Verbal	SAT Quant.	GPA
1.	623	509	2.6
2.	454	471	2.3
3.	643	700	2.4
4.	585	719	3.0
5.	719	710	3.1
6.	693	643	2.9
7.	571	665	3.1
8.	646	719	3.3
9.	613	693	2.3
10.	655	701	3.3

Fig. 6-3. Data for statistical analysis.

The Data

This data set shows three scores for each of 10 students at a certain university on the east coast. The first two scores for each student are from the Verbal Section and the Quantitative Section of the Scholastic Aptitude Test. The third score is the student's freshman grade point average.

THE READ COMMAND

To analyze this data, you must first get it into your *workspace*, which is the buffer used by Minitab. To do that, enter the Minitab Mode again, and then give the command **READ C1 C2 C3**. This will move you into the Minitab READ Mode, which is like the EDT INSERT Mode. The parameters C1, C2, and C3 tell the system that you are going to type in three columns of data (one for each score). The prompt for the READ Mode is DATA>, telling you that the system is ready to receive more data. Type in the data now, as shown in Fig. 6-4.

```
$ MINITAB
MINITAB RELEASE 82.1 *** COPYRIGHT - PENN STATE UNIVERSITY
MAY 23, 1985 *** THE UNIVERSITY OF RICHMOND *ACADEMIC COMPUTING
STORAGE AVAILABLE 937416
MTB > READ C1 C2 C3

DATA> 623 509 2.6
DATA> 454 471 2.3
DATA> 643 700 2.4
DATA> 585 719 3.0
DATA> 719 710 3.1
DATA> 693 643 2.9
DATA> 571 665 3.1
DATA> 646 719 3.3
DATA> 613 693 2.3
DATA> 655 701 3.3
DATA> <CTRL/Z> Exit
  10 ROWS READ
MTB >
```

Fig. 6-4. Executing the READ command to read data into the workspace.

The READ command tells Minitab that you want to "read" some data into your workspace, which is organized by columns. The data you want to analyze is for three variables, so you use three columns, called C1, C2, and C3. After the data for the tenth student is typed, press <CTRL/Z> to exit from the READ Mode. This puts you back into the Minitab Mode.

Column Labels

After entering new data into your workspace, it is always a good idea to label your columns with English words. This is done with the NAME command. Follow the example in Fig. 6-5. The NAME command assigns the specified character string to the column indicated.

```
MTB > NAME C1 'VERBAL'
MTB > NAME C2 'MATH'
MTB > NAME C3 'GPA'
MTB >
```

Fig. 6-5. Executing the NAME command to label the data.

Examining Your Workspace

Now, give the INFORMATION command to get a summary of what is in your workspace. The result should look like Fig. 6-6.

```
MTB > INFORMATION

COLUMN     NAME       COUNT
C1         VERBAL     10
C2         MATH       10
C3         GPA        10

CONSTANTS USED: NONE

MTB >
```

Fig. 6-6. Executing the INFORMATION command to see the status of the current contents of the workspace.

The INFORMATION command displays which columns are in use in your workspace, what they have been named, and how many data values are in each one. It also reports on your use of constants, which will be discussed later.

Use the PRINT command to get a complete display of your data. It should look like Fig. 6-7. The PRINT command displays the columns specified. These columns can be specified by range, by sequence, or by name. Also, if listed sequentially, the order is irrelevant, and most other characters are ignored.

```
MTB > PRINT C1-C3
 ROW  VERBAL    MATH     GPA

   1      623      509     2.6
   2      454      471     2.3
   3      643      700     2.4
   4      585      719     3.0
   5      719      710     3.1
   6      693      643     2.9
   7      571      665     3.1
   8      646      719     3.3
   9      613      693     2.3
  10      655      701     3.3

MTB >
```

Fig. 6-7. Executing the PRINT command to display the contents of the workspace.

Thus, the following column specifications are equivalent:

```
C1-C3
C1, C2, C3
'VERBAL', 'MATH', 'GPA'
C3, 'MATH', 'VERBAL'
C2 and 'GPA' WITH C1
```

Column names must be enclosed with apostrophes to distinguish them from other (ignored) words. Notice the similarity between the PRINT command in the Minitab Mode and the TYPE WHOLE command in the EDT Editing Mode.

Simple Statistics

Now that you have data in your workspace, you are ready to do some statistical analysis on it. One of the simplest statistics is the ***arithmetic mean.*** This is just the ordinary average: the sum of a set of numbers divided by the number in the set. You can obtain the mean of all the numbers in any column of your workspace by using the MEAN command, as shown in Fig. 6-8. Other

```
MTB > MEAN 'VERBAL'
   MEAN        =       620.20
MTB > MEAN 'MATH'
   MEAN        =       653.00
MTB > MEAN 'GPA'
   MEAN        =       2.8300
MTB >
MTB > PRINT C1-C3
 ROW   VERBAL     MATH     GPA

   1      623      509     2.6
   2      454      471     2.3
   3      643      700     2.4
   4      585      719     3.0
   5      719      710     3.1
   6      693      643     2.9
   7      571      665     3.1
   8      646      719     3.3
   9      613      693     2.3
  10      655      701     3.3

MTB >
```

Fig. 6-8. Executing the MEAN command to compute the arithmetic mean of each column of data.

common statistics, such as the ***median*** (the middle number) and the ***standard deviation*** (a measurement of how spread out the data is), are just as easy to obtain. See Fig. 6-9.

The DESCRIBE command lists the most common descriptive statistics for the variables you specify, as shown in Fig. 6-10. Here, N is the number of pieces

```
MTB > MEDIAN 'VERBAL'
   MEDIAN =      633.00
MTB > STDEV 'GPA'
   ST.DEV. =     0.39735
MTB > STDEV 'MATH'
   ST.DEV. =     89.553
MTB > MEDIAN 'GPA'
   MEDIAN =      2.9500
MTB >
```

Fig. 6-9. Executing the MEDIAN and STDEV commands to compute the median and standard deviation of each column of data.

```
MTB > DESCRIBE 'VERBAL'

               N      MEAN    MEDIAN   TRMEAN     STDEV    SEMEAN
VERBAL        10     620.2     633.0    628.6      73.6      23.3

              MIN       MAX        Q1       Q3
VERBAL      454.0     719.0     581.5    664.5

MTB >
```

Fig. 6-10. Executing the DESCRIBE command to compute various statistics for one column of data.

of data, TRMEAN is the "trimmed mean" (the mean of the middle 90% of the data), SEMEAN is the mean divided by the standard deviation (STDEV), and Q1 and Q3 are the first and third quartiles.

If the numbers displayed on your screen are not the same as those shown here, then you probably entered some of the data incorrectly.

The Histogram

Minitab also displays data graphically. One of the most common such displays is the *histogram*, which is a graphical representation of the frequencies of the data in each of several groups. When the data consists of measurements (such as SAT scores), the groups are simply adjacent intervals. Give the HISTOGRAM command as in Fig. 6-11. Each group in this example spans a range of 40 points. The last line, for example, reveals that there was one score in the range 700 to 740. Histograms are usually used on larger sets of data.

```
MTB > HISTOGRAM 'VERBAL'

Histogram of VERBAL   N = 10

Midpoint   Count
     440       1  *
     480       0
     520       0
     560       1  *
     600       2  **
     640       4  ****
     680       1  *
     720       1  *

MTB >
```

Fig. 6-11. Executing the HISTOGRAM command to display a histogram of one column of data.

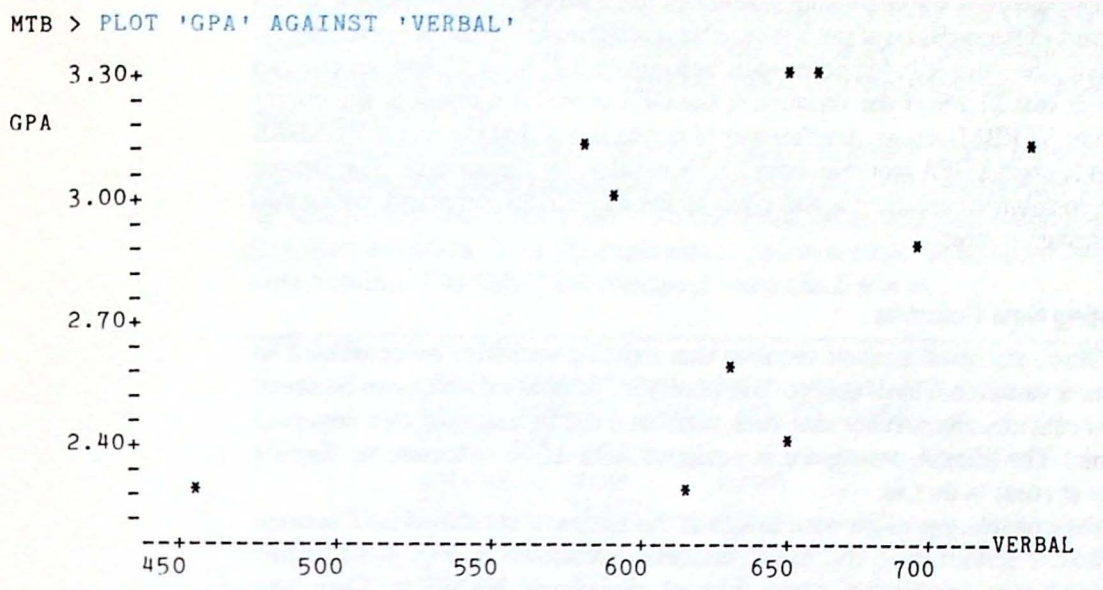

Fig. 6-12. Executing the PLOT command to plot a scatter diagram of the relationship between two columns of data.

Plotting Data

One very common statistical activity is the effort to detect relationships between variables. The simplest kind of relationship is a linear relationship, which is fairly easy to detect when one variable is plotted against another. The purpose of the Scholastic Aptitude Examinations is to predict success in college. Thus, student's GPA scores should depend upon their SAT scores. Use the PLOT command, as shown in Fig. 6-12, to look for such dependence in the sample of 10 freshmen.

Evidently, the relationship is not very strong. A better measure of the relationship is obtained from the CORRELATE command, shown in Fig. 6-13.

```
MTB > CORRELATE 'GPA' WITH 'VERBAL'

Correlation of GPA and VERBAL = 0.465

MTB >
```

Fig. 6-13. Executing the CORRELATE command to compute the correlation coefficient between two columns of data.

The Correlation Coefficient

This command computes the *correlation coefficient* between two columns of data. It is always a number between -1.0 and 1.0; the closer its value is to zero, the lower the correlation. The value 0.465, therefore, shows that GPA and VERBAL are hardly correlated at all.

The square of the correlation coefficient has a simple interpretation: it tells what part of the variation of the Y=variable is attributable to the X=variable. In this case, the value is 0.465 squared, which equals 0.216, or 21.6%, so you can conclude that 21.6% of the variation of the GPA scores is a result of the corresponding VERBAL scores. Another way of saying this is that the use of VERBAL scores to predict GPA scores is about 21.6% reliable, for this sample. The square of the correlation coefficient is also given by the REGRESS command, which you will encounter later.

Creating New Columns

Often, statistical analysis requires that existing variables be combined to form new variables. This is easy to do in Minitab. The new variables can be saved in new columns. (Remember that each variable must be assigned to a separate column.) The Minitab workspace is equipped with 1000 columns, so there's plenty of room to do this.

For example, you might want to look at the students' combined SAT scores: VERBAL + MATH. Use the ADD command, as shown in Fig. 6-14. After creating a new variable (i.e. a new column), you should NAME it. Then, the INFORMATION command is used to show the current state of the workspace,

```
MTB > ADD 'VERBAL' AND 'MATH' INTO C4
MTB > NAME C4 'SUM'
MTB > INFORMATION

COLUMN     NAME      COUNT
C1         VERBAL    10
C2         MATH      10
C3         GPA       10
C4         SUM       10

CONSTANTS USED: NONE

MTB > PRINT C1-C4
 ROW  VERBAL    MATH     GPA     SUM
   1     623     509     2.6    1132
   2     454     471     2.3     925
   3     643     700     2.4    1143
   4     585     719     3.0    1304
   5     719     710     3.1    1429
   6     693     643     2.9    1336
   7     571     665     3.1    1236
   8     646     719     3.3    1365
   9     613     693     2.3    1306
  10     655     701     3.3    1356

MTB > CORRELATE 'GPA' WITH 'SUM'

Correlation of GPA and SUM = .560

MTB >
```

Fig. 6-14. Executing the ADD command to add the corresponding data from two columns.

and the PRINT command to give all of the details. Finally, the CORRELATE command shows that, for this sample, the GPA is better correlated with the combined SAT scores than with just the VERBAL score. (However, the correlation is still very low.)

A common statistical transformation of a distribution of data is the *Z-score* distribution, also called the normal distribution. This is obtained for each number by subtracting the mean value from the actual value, and then dividing by the standard deviation. This "normalization" makes it easier to compare different distributions. The CENTER command computes Z-scores.

```
MTB > CENTER 'SUM' 'GPA' INTO C5, C6
MTB > NAME C5 'ZSUM' AND C6 'ZGPA'
MTB > INFO

COLUMN     NAME       COUNT
C1         VERBAL     10
C2         MATH       10
C3         GPA        10
C4         SUM        10
C5         ZSUM       10
C6         ZGPA       10

CONSTANTS USED: NONE

MTB > PRINT C4 C5 C3 C6
 ROW     SUM       ZSUM     GPA       ZGPA

   1    1132   -0.96432     2.6   -0.57833
   2     925   -2.37801     2.3   -1.33383
   3    1343    0.47670     2.4   -1.08216
   4    1304    0.21035     3.0    0.42783
   5    1429    1.06403     3.1    0.67950
   6    1336    0.42889     2.9    0.17617
   7    1236   -0.25406     3.1    0.67950
   8    1365    0.62694     3.3    1.18283
   9    1306    0.22401     2.3   -1.33383
  10    1356    0.56548     3.3    1.18283

MTB >
```

Fig. 6-15. Executing the CENTER command to compute Z scores for two columns.

In this example, the CENTER command is used to standardize both SUM and GPA. After naming these new variables and checking INFORMATION, the PRINT command shows their distributions for the given sample.

Theoretically, the mean Z-score is exactly 0.0, and the standard deviation is exactly 1.0. But on computers, round-off error always prevents exactness. If the data is normally distributed (rendering a histogram bell-shaped), then about 68% of the Z-scores will be between −1.0 and 1.0, 95% will be in the range −2.0 to 2.0, 99% will be in the range −3.0 to 3.0, and the distribution will be symmetric about 0.0. This sample distribution does not appear to be normal. You can see, by comparing the ZSUM and ZGPA scores, that students 2, 7, 8, and 10 are doing

considerably better than predicted by their SAT scores (relative to this small population), and that students 3 and 9 are doing worse.

THE SAVE COMMAND

Now that you have all this data, you should SAVE it for future reference. Give the SAVE command, as shown in Fig. 6-16. The parameter passed to the SAVE command becomes the filename of the file in which the data is saved. Here, the filename SCORES has been chosen. (Notice that it must be enclosed in apostrophes.) Minitab will automatically add the filetype MTW.

```
MTB > SAVE 'SCORES'
MTB > STOP
*** Minitab Release 5.1 *** Minitab, Inc. ***
Storage available 896430


$ DIR SCORES.*

Directory SYS$USER:[REAGONRON]

SCORE.MTW;1

Total of 1 file.
$
```

Fig. 6-16. Executing the SAVE command to save the contents of the workspace in a (non-ASCII) file.

To see that a new file called SCORES.MTW has actually been created, give the STOP command to exit from Minitab and check your directory, as shown in Fig. 6-16.

Unfortunately, the data in the file is readable only from within Minitab, because this file is not an ASCII text file. If you try to display it using the TYPE command in DCL Mode (as in Fig. 6-17), it appears as garbage.

```
$ T SCORES.MTW
Worksheet stored by Minitab release  5.1 VX 1986 727

z2=A~

@cD E@E@3E@-E@E@!EE@@#E@

HB3
~DkD/E3E@1E E@&E@3E@-E@/E@
```

Fig. 6-17. Attempting to TYPE a (non-ASCII) Minitab MTH file.

ASCII and Non-ASCII Files

In terms of human readability, external files are classified as being either *ASCII* or *non-ASCII*. An ASCII file is one which contains only ASCII characters (see Appendix J). All the files created by the EDT Editor are ASCII files. When they are displayed by the TYPE command, they can be read by humans. A non-ASCII file can be translated by the operating system into data which is readable by humans, but when it is displayed directly by the TYPE command, it looks like a random sequence of characters. That is because the stored bits are not meant to be interpreted as single-byte ASCII characters. The object modules produced by compilers and the executable images produced by the Linker (see Chapter 5) are also non-ASCII files.

To save your Minitab data in an ASCII file, return to the Minitab Mode, RETRIEVE your data from the non-ASCII file SCORES.MTW, check your workspace with the INFORMATION command, and then execute the WRITE command, as shown in Fig. 6-18.

Unlike the SAVE command, the WRITE command requires that you specify which columns are to be copied into the new file. In this example, all six columns were copied into the file, but you could have copied any subset of those six.

Now, go back to DCL Mode and see what is in this new file. It should look like Fig. 6-19. The DIRECTORY command reveals that a new file named SCORES.DAT has been added to the directory. The file type, DAT, is automatically assigned by Minitab; it stands for DATA. The TYPE command shows that the file is an ASCII file that contains all 60 numbers from the Minitab workspace. The data is arranged in ten rows and six columns, just as you would type it if you were entering the data by means of the READ command in Minitab. Note,

```
$ MINITAB
MINITAB RELEASE 5.1 *** COPYRIGHT - MINITAB, INC. 1985
U.S. FEDERAL GOVERNMENT USERS SEE HELP FGU
JULY 28, 1986 *** University of Richmond
STORAGE AVAILABLE 896430

 Type NEWS for information on new features in Minitab Release #5.1

MTB > RETRIEVE 'SCORES'
MTB > INFO

COLUMN     NAME       COUNT
C1         VERBAL        10
C2         MATH          10
C3         GPA           10
C4         SUM           10
C5         ZSUM          10
C6         ZGPA          10

CONSTANTS USED: NONE

MTB > WRITE 'SCORES' C1-C6
MTB >
```

Fig. 6-18. Executing the RETRIEVE and WRITE commands.

```
MTB > STOP
*** Minitab Release  5.1 *** Minitab, Inc. ***
Storage available 896430


$ DIR SCORES.*

Directory SYS$USER:[REAGANRON]

SCORES.DAT;1          SCORES.MTW;1

Total of 2 files.
$ T SCORES.DAT
  623 509 2.6 1132 -0.96432 -0.57833
  454 471 2.3 925  -2.37801 -1.33383
  643 700 2.4 1343  0.47670 -1.08216
  585 719 3.0 1304  0.21035  0.42783
  719 710 3.1 1429  1.06403  0.67950
  693 643 2.9 1336  0.42889  0.17617
  571 665 3.1 1236 -0.25406  0.67950
  646 719 3.3 1365  0.62694  1.18283
  613 693 2.3 1306  0.22401 -1.33383
  655 701 3.3 1356  0.56548  1.18283
$
```

Fig. 6-19. Displaying a (ASCII) Minitab DAT file.

however, that the column names (VERBAL, GPA, etc.) are not saved in this data file.

Default Files

You have seen that there are two ways to copy your data from your Minitab workspace into a file in your DCL directory: the WRITE command and the SAVE command. You have also seen two ways to get data into your workspace: the READ command and the RETRIEVE command. Now, you shall see that these two commands are the *inverse* commands of the first two. Each reverses the action of its inverse.

The READ command was first used to read data in from the keyboard. The READ procedure uses the keyboard as the default input file, which means that, unless you explicitly specify some file from which the data is to be read, the system will expect you to type the data in at the keyboard. Most statistical analysis, however, is done on large data sets; hundreds, or even thousands, of numbers, instead of just 20 or 60. It would be very difficult (and tedious) to type in that many numbers at the keyboard. Consequently, nearly all statistical data is stored in separate ASCII files before any analysis is carried out on it. So, to be realistic, you should have Minitab read data from an ASCII file. The SCORES.DAT file, which Minitab created with the WRITE command, is such a file, and its data is arranged just the way the Minitab READ command expects; each row represents a different sample element, and each column represents a separate variable.

Return to the Minitab Mode and READ the data from the SCORES.DAT file, as shown in Fig. 6-20.

```
$ MI
MINITAB RELEASE 5.1 *** COPYRIGHT - MINITAB, INC. 1985
U.S FEDERAL GOVERNMENT USERS SEE HELP FGU
JULY 28, 1986 *** University of Richmond
STORAGE AVAILABLE 896430

 Type NEWS for information on new features in Minitab Release #5.1

MTB > INFO
 * ALL COLUMNS EMPTY AND UNNAMED

CONSTANTS USED: NONE

MTB > READ 'SCORES' C1-C6
     10 ROWS READ
ROW      C1      C2      C3      C4          C5          C6

   1    623     509     2.6    1132    -0.96432    -0.57833
   2    454     471     2.3     925    -2.37801    -1.33383
   3    643     700     2.4    1343     0.47670    -1.08216
   4    585     719     3.0    1304     0.21035     0.42783

MTB > INFO

COLUMN     NAME      COUNT
C1                      10
C2                      10
C3                      10
C4                      10
C5                      10
C6                      10

CONSTANTS USED: NONE

MTB >
```

Fig. 6-20. Executing the READ command on an existing DAT file. The filename 'SCORES' tells Minitab to read the data from the file SCORES.DAT instead of from the keyboard.

The READ command may be used with or without the name of a file. If no file is mentioned, the data must be typed in at the keyboard; otherwise, it is read directly from the file specified. If no file type is specified, Minitab assumes that the type is DAT (that is, DAT is the default). Note that only the first four rows of the data are displayed during the READ command, but Minitab states that 10 rows were read.

The WRITE and READ commands work almost as well as the SAVE and RESTORE commands. The advantages of the latter pair is that they also preserve the column names, and take less time to execute. The disadvantage is that they work only with non-ASCII files, so that the only way to ascertain what is in your SCORES.MTW file is to return to the Minitab Mode, RESTORE it to your workspace, and then PRINT it. On the other hand, you can see exactly what is in your SCORES.DAT file simply by executing the TYPE command in the DCL Mode.

The fact that the SAVE/RESTORE option preserves the column names is important. If you don't know what the data represents, then it has no meaning. The advantage of the WRITE/READ option, however, is also important; it allows you to create and edit your data files in the EDT editor.

RETRIEVING DATA

Minitab allows you to rearrange your data easily. RETRIEVE your SCORES file and check your workspace. Follow the example in Fig. 6-21 to see how the ERASE, RANK, and SORT commands work. Note that the previous

```
MTB > RETR 'SCORES'
MTB > INFO

COLUMN     NAME     COUNT
C1         VERBAL      10
C2         MATH        10
C3         GPA         10
C4         SUM         10
C5         ZSUM        10
C6         ZGPA        10

CONSTANTS USED: NONE

MTB > ERASE C2, C4-C6
MTB > INFO

COLUMN     NAME     COUNT
C1         VERBAL      10
C3         GPA         10

CONSTANTS USED: NONE

MTB > RANK 'VERBAL' INTO C2
MTB > PRINT C1-C2
 ROW  VERBAL    C2

   1     623     5
   2     454     1
   3     643     6
   4     585     3
   5     719    10
   6     693     9
   7     571     2
   8     646     7
   9     613     4
  10     655     8

MTB > SORT 'VERBAL' CARRYING 'GPA' INTO C5-C6
MTB > PRINT C1 C3 C5 C6
 ROW  VERBAL   GPA      C5      C6

   1    623    2.6     454     2.3
   2    454    2.3     571     3.1
   3    643    2.4     585     3.0
   4    585    3.0     613     2.3
   5    719    3.1     623     2.6
   6    693    2.9     643     2.4
   7    571    3.1     646     3.3
   8    646    3.3     655     3.3
   9    613    2.3     693     2.9
  10    655    3.3     719     3.1

MTB >
```

Fig. 6-21. Executing the ERASE, RANK, and SORT commands.

content of your workspace has been replaced by the data in SCORES.MTW. Of course, the numbers are the same, but remember that the RETRIEVE command recalls the names of the variables, too.

The ERASE command does just what you would expect it to: it removes the columns specified. The RANK command computes the relative ranks of the numbers in a specified column, and then writes those ranks into the new column (C2 in this case). For example, the PRINT command shows that Student 1, whose VERBAL score was 623, ranks fifth, and that Student 2, whose score was 454, ranks first. Note that the ranking is from lowest number to highest number, regardless of the meaning of the numbers.

The SORT command rearranges the data into increasing order. In this example, you are sorting on the VERBAL scores, carrying the GPA scores along so that you can tell which GPA score goes with which VERBAL score. For example, fifth-ranked Student 1 has both VERBAL and GPA scores moved to row 5 in columns C5 and C6.

So far, all of the numbers you have worked with have been in columns. Minitab also allows you to use individual numbers, stored in separate individual variables. These variables are called constants in Minitab, and are labeled K1, K2, etc. Two special constants are automatically defined: K999 equals 2.71828, and K1000 stands for 3.14159. These are approximations of the mathematical constants e (2.718281828459...) and pi (3.141592653589...).

Array Processing

New constants, and complete columns, can be combined in Minitab by means of the LET command. This allows you to do rather powerful *array processing*. See Fig. 6-22 for examples. This example also illustrates some new arithmetic commands. The COUNT command puts the number of values held in column C3 into the constant K6. The SUM command puts the sum of the values of column C4 into the constant K7. The MAXIMUM command puts the largest of the values of column C3 into constant K8, while the ADD command puts the sum

```
MTB > LET C2 = C5
MTB > ERASE C4-C6
MTB > LET K1 = 1/2 + 3/K1000 - K999
MTB > LET K2 = 2**ROUND(K1)
MTB > LET C4 = K1000*'GPA'**2
MTB > LET K3 = MEAN (C4)
MTB > LET C5 = LOGTEN(C1-K3)
MTB > LET K4 = MEDIAN(C2)
MTB > LET C6 = (C1 - K4)**4
MTB > LET C7 = LOGE(ABS(C4 - K3)) - LOGE(STDEV(C4))
MTB > LET K5 = SIGN(K2)*SQRT(ABSO*K2))
MTB > COUNT C3 INTO K6
MTB > SUM C4 INTO K7
MTB > MAXIMUM C3 INTO K8
MTB > ADD K1-K8 INTO K9
MTB > PRINT K1-K9
```

Fig. 6-22. Manipulating constants in Minitab.

of constants K1 through K8 into constant K9. In the last step, the PRINT command is used to display constants as well as values.

Simulation

One of the most important applications of computers is *simulation.* This is the process of modeling a real phenomenon (like the weather) by means of variables whose values change with time according to the underlying principles of the system being modeled. An essential part of any simulation is a randomness factor which affects the model the same way chance affects the phenomenon being modeled. Such randomness is usually generated by a *random number function.* Minitab provides a wide variety of random number functions.

A random number function is classified according to the probability distribution of its output, and the parameters of that distribution. For example, if the distribution is UNIFORM and the parameters are 2.7 and 5.2, then the random numbers produced would tend to be uniformly distributed in the range from 2.7 to 5.2.

Figure 6-23 shows how to use the RANDOM function to generate 16 numbers from a BINOMIAL distribution with parameters n = 100 and p = 0.65. These 16 numbers would simulate the outcomes of 16 polls, each of 100 people selected at random. Each number represents the number of people out of 100 who said "yes" to a question to which, ordinarily, 65% of all the people would say "yes." For example, suppose 65% of all the people in the U.S. approved of President Reagan's foreign policy. Then, if 100 people were selected at random and asked if they supported his policy, the number who said "yes" would tend to be around 65. The 16 numbers generated here are all around 65.

The RANDOM command uses a *subcommand,* which is indicated by ending the command line with a semicolon (;). This causes Minitab to respond with the SUBC> prompt. With the RANDOM command, the subcommand gives the distribution (BINOMIAL, INTEGERS, etc.) and its parameters. Notice that the subcommand must end with a period.

SELF-TEST EXERCISES

1. RETRIEVE your SCORES file, and then enter the command:

   REGRESS 'GPA' on 2 COLUMNS 'VERBAL' and 'MATH'

   This command will produce a lot of information. Try to interpret the "regression equation."

```
MTB > RANDOM 16 C1;
SUBC> BINOMIAL 100 .65.
MTB > PRINT C1
C1
   70    73    61    60    67    61    68    62    70    71    61    56
   66    64    63    61
```

Fig. 6-23. Executing the RANDOM command to generate random numbers. The subcommand BINOMIAL 100 .65. causes the 16 numbers to be distributed according to the binomial distribution with parameters n = 100 and p = .65.

2. The SET command, which is used to enter data into one column, has several "short-cut" features. Execute the commands shown in Fig. 6-24, and then PRINT your results to see their effects:

```
MTB > SET C1
DATA> 3(1,2,3,4,5)
MTB > SET C2
DATA> (11,22,33)5
MTB > SET C3
DATA> 77:44/8
MTB > PRINT C1-C3
```

Fig. 6-24. Executing the SET command to put data into columns.

3. Investigate the Minitab HELP facility. First, give the HELP command. Then, try HELP HELP. Next, see what HELP OVERVIEW does. Finally, try some of the specific HELP OVERVIEW chapters, especially HELP OVERVIEW 1, HELP OVERVIEW 2, HELP OVERVIEW 3, HELP OVERVIEW 4, and HELP OVERVIEW 7.
4. Use the HELP RANDOM command to investigate the RANDOM command, which is used to generate random numbers with a given distribution, as discussed in the last section of this chapter. The distribution and its parameters are specified in the subcommand. Use the RANDOM command with the subcommand INTEGERS to generate a sequence of 10 integers into C1 and 10 into C2, all of which are uniformly distributed in the range from one to six. Then, use this data to produce 10 numbers in C3 which simulate 10 tosses of a pair of balanced dice. Each toss results in a number in the range from two to 12 (which are not uniformly distributed).
5. If you have studied the Trapezoidal Rule for numerical integration, use Minitab to apply the method to integrate the function f(X) = 6/SQRT(1−X*X) on the interval [0,0.5]. Your answer should be close to pi.
6. Use the HELP STORE command to investigate the STORE command. This is used to create a macro, i.e., a file containing a sequence of commands which are sequentially executed automatically. Use it to create a macro which uses only LET commands and has the same effect as the CENTER command.

Chapter 7

Running a BASIC Program

It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e. express it as an algorithm.

Donald E. Knuth

The BASIC programming language was developed by John G. Kemeny and Thomas E. Kurtz at Dartmouth College in 1964 to help students learn to use computers. (The word "BASIC" is an acronym for Beginner's All-purpose Symbolic Instructional Code.) Since then, it has become the most popular programming language for microcomputers.

THE VERSIONS OF BASIC

Many dialects, or extensions, of the BASIC language are in use today. The version available on the VAX is called VAX-11 BASIC. It is a high-powered extension of standard BASIC, developed by the Digital Equipment Corporation.

In this chapter, you will learn how to create and run simple BASIC programs on the VAX. Readers who are familiar with the BASIC programming language should be able to quickly extend this elementary skill to full use of the VAX's interactive BASIC programming mode.

Compiled and Interpreted BASIC

There are two kinds of BASIC programming environments available on the VAX: *compiled* BASIC, and *interpreted* BASIC. Compiled BASIC works much like compiled Pascal. After the complete program has been written in a file (using the EDT editor), the BASIC compiler is invoked (with the BASIC command) to translate the source code into machine language. Then, the VAX Linker is used to convert the resulting object module into an executable image, which can be run repeatedly.

Interpreted BASIC is translated and run line-by-line every time the RUN command is given. The program which performs this activity is called the BASIC *interpreter*. This type of BASIC is far less efficient than compiled BASIC, because every line must be translated all over again every time the program is executed. It is easier for beginners to use, however, because the translating and linking is all done automatically whenever the user types RUN. Because BASIC is used mostly by beginners, it is far more common in its interpreted form. This is the form used on microcomputers.

The VAX BASIC Mode

The VAX BASIC Programming Environment appears to the user much like the ordinary environment on a microcomputer. From now on, this environment will be called the BASIC Mode. Fig. 7-1 shows how this mode fits into the VAX system. To enter the BASIC Mode, give the DCL command, BASIC as shown in Fig. 7-2.

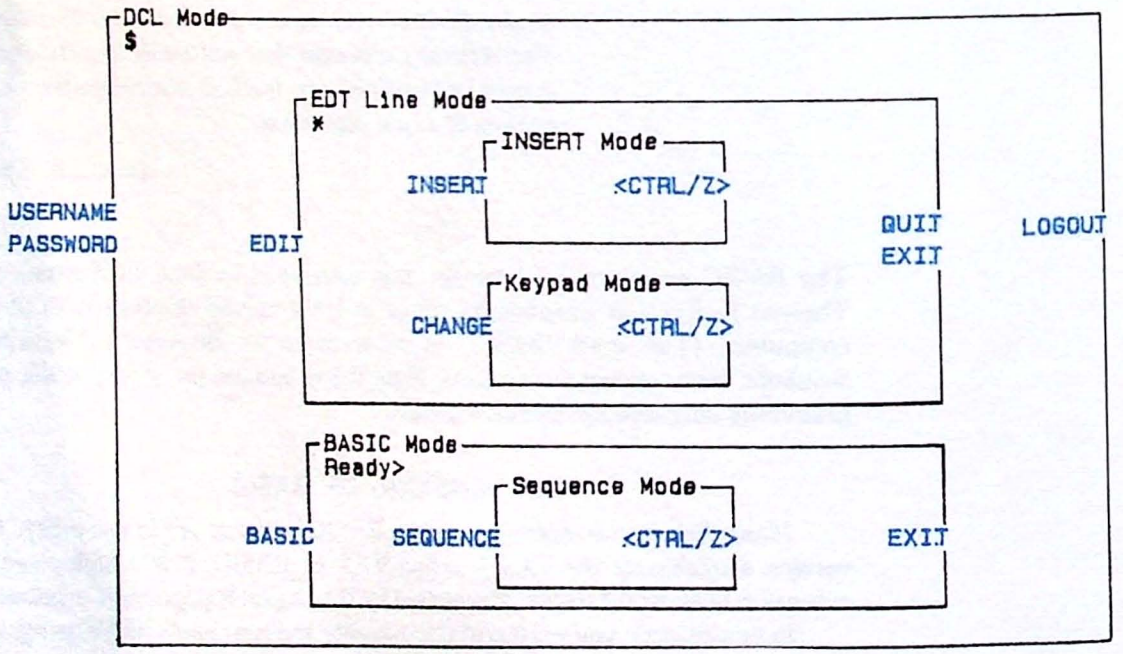

Fig. 7-1. The DCL Mode, the EDT Line Mode, the INSERT Mode, the Keypad Mode, the BASIC Mode, and the Sequence Mode.

```
$ BASIC
VAX BASIC V2.4
Ready
EXIT
```

Fig. 7-2. Executing the BASIC and EXIT commands to enter and exit the BASIC Mode.

In the BASIC mode, the prompt is **Ready<RET><RET>**; that is, the system displays "Ready," and then moves the cursor down two lines. The BASIC Mode command which returns you to DCL Mode is EXIT. Use these commands to practice entering and exiting the BASIC Mode.

As with any mode on the VAX, you can always leave abruptly and return to the DCL Mode by pressing <CTRL/Y>. Interrupting the mode this way, however, is usually unwise. One reason is that the contents of your current buffer will not be saved when you press <CTRL/Y>. Also, like the EDT Line Mode and the Minitab Mode, the BASIC Mode has its own submode. This mode is called the Sequence Mode, and is entered by the SEQUENCE command. Like other submodes, the <CTRL/Z> command returns you to the previous mode.

THE BASIC PROGRAM

Enter the SEQUENCE Mode, press <RET> a few times, and then press <CTRL/Z>. The purpose of this mode is obvious: it provides line numbers for your BASIC program. Repeat this step to see that the sequencing continues where it left off, instead of starting over at 100. You are, in effect, writing a BASIC program consisting of empty statements.

Unlike programs written in other high-level languages like Pascal and C, BASIC programs require a line number for each statement. In VAX BASIC, you may type the line numbers yourself, or you may use the Sequence Mode to have it done for you automatically. Notice that, in this mode, the numbering begins with 100 and increments by 10. You may use any numbers you want as long as they are increasing and are between 1 and 32767, inclusive.

To write a simple BASIC program to compute and print the square root of a

```
Ready

SEQUENCE
100     <RET>
110     <RET>
120     <RET>
130     <RET>
140     <CTRL/Z>Exit
Ready
```

Fig. 7-3. Executing the SEQUENCE command to generate line numbers.

given number, follow the example given in Fig. 7-4. You now have a complete BASIC program, named SQUAREROOT, in your buffer. The buffer will hold only one program, so it will be called the current program. You may view the complete current program at any time by giving the LIST command. The LIST command prints the program name, the date and time, and a complete listing of the current program in order of its line numbers.

```
Ready

NEW SQUAREROOT
Ready

SEQUENCE
100    INPUT X
110    LET Y = SQRT(X)
120    PRINT Y
130    END
140    <CTRL/Z>Exit
Ready

LIST
SQUAREROOT     25-JUL-1986 09:07

100      INPUT X
110      LET Y = SQRT(X)
120      PRINT Y
130      END

Ready
```

Fig. 7-4. Writing and listing a simple BASIC program.

Running the Program

RUN your program a few times, as shown in Fig. 7-5. The BASIC Mode provides the prompt ? when the program that is running requires input. If you enter a negative number in this example, the program crashes, and the system displays several diagnostic error messages. The first message tells why the program crashed. The third message tells which line was being executed when the crash occurred. Recall that interpreted BASIC translates and executes each line separately, so on this last run, line 120 did not even get translated.

Debugging

Fix your program, as shown in Fig. 7-6, so that it won't crash. Then run it again, as in Fig. 7-7.

Editing

You can add lines to your program any time. Just type the appropriate line number, followed by the program statement that you want to add as in Fig. 7-8.

```
Ready

RUN
SQUAREROOT    25-JUL-1986 09:08

? 49
 7
Ready

RUN
SQUAREROOT    25-JUL-1986 09:09

? 59
 7.68115
Ready

RUN
SQUAREROOT    25-JUL-1986 09:09

? -9
%BAS-F-IMASQUROD, Imaginary square roots
-BAS-I-USEPC_PSL, at user PC=000C0578, PSL=03C00000
-BAS-I-FROLINMOD, from line 110 in module SQUAREROOT
-MTH-F-SQUROONEG, square root of negative value
  user PC 0017E95C
Ready

Ready
```

Fig. 7-5. Running a BASIC program and encountering a run-time error.

```
Ready

LIST
SQUAREROOT    25-JUL-1986 09:14

100      INPUT X
110      LET Y = SQRT(X)
120      PRINT Y
130      END

Ready

110      IF X > 0 THEN LET Y = SQRT(X)
LIST
SQUAREROOT    25-JUL-1986 09:14

100      INPUT X
110      IF X > 0 THEN LET Y = SQRT(X)
120      PRINT Y
130      END

Ready
```

Fig. 7-6. Debugging the program.

```
Ready

RUN
SQUAREROOT     25-JUL-1986 09:08

? 49
 7
Ready

RUN
SQUAREROOT     25-JUL-1986 09:09

? 59
 7.68115
Ready

RUN
SQUAREROOT     25-JUL-1986 09:09

? -9
 0
Ready
```

Fig. 7-7. Running the corrected program.

In this example, the statement IF X < 0 THEN GOTO 100 is added at line 105. The line number 105 is used because this new statement is to be put between the statements at lines 100 and 110. (You could have used 103 or any other number between 101 and 109, inclusive.)

The GOTO statement is used frequently in BASIC programs. It causes the program control to be diverted to the line whose number is specified; in this case,

```
Ready

105      IF X < 0 THEN GOTO 100
110      LET Y = SQRT(X)
LIST
SQUAREROOT     25-JUL-1986 09:17

100      INPUT X
105      IF X < 0 THEN GOTO 100
110      LET Y = SQRT(X)
120      PRINT Y
130      END

Ready

RUN
SQUAREROOT     25-JUL-1986 09:18

? -9
? -385
? -1
? 65536
 256
Ready
```

Fig. 7-8. Making the program more "user-friendly".

line 100. Line 100 will be repeated as long as the values typed in for X are negative. This is one way to solve the problem, but not the best way.

You can delete lines from a BASIC program simply by typing their line numbers, each followed immediately by <RET>. Follow Fig. 7-9 to delete line 110, and then change line 120. These changes really do improve the program. Now, it is crashproof, 20% shorter, and uses half as many variables.

```
Ready

110
LIST
SQUAREROOT    25-JUL-1986 09:22

100     INPUT X
105     IF X < 0 THEN GOTO 100
120     PRINT Y
130     END

Ready

120     PRINT SQRT(X)
Ready

LIST
SQUAREROOT    25-JUL-1986 09:23

100     INPUT X
105     IF X < 0 THEN GOTO 100
120     PRINT SQRT(X)
130     END

RUN
SQUAREROOT    25-JUL-1986 09:18

? -65536
? 65536
 256
Ready
```

Fig. 7-9. Further modifications. Line 110 is deleted and line 120 is replaced.

THE SAVE COMMAND

So far, all of your activity in this chapter has been within the BASIC Mode. You can leave that mode any time by giving the EXIT command, but the program you have been working on exists only in your BASIC Mode buffer, and that buffer will disappear as soon as you EXIT. So, if you were to EXIT now, your program would be lost.

The SAVE command will prevent your program from being lost. Most VAX BASIC programmers regularly enter this command, as a precaution. Do so now.

The SAVE command creates a new file in your directory, gives it the filename you used when you gave the NEW command, gives it the file type BAS, and then copies the current contents of your BASIC Mode buffer into this new file. The DCL TYPE command verifies that your program has been saved. Notice that the line numbers are part of the file.

```
Ready

SAVE
Ready

LIST
SQUAREROOT    25-JUL-1986 09:27

100      INPUT X
105      IF X < 0 THEN GOTO 100
120      PRINT SQRT(X)
130      END

EXIT
$ DIR *.BAS

Directory

SQUAREROOT.BAS;1

Total of 1 file.
$ T SQUAREROOT.BAS
100      INPUT X
105      IF X < 0 THEN GOTO 100
120      PRINT SQRT(X)
130      END
```

Fig. 7-10. Executing the SAVE command to write the current program into a permanent BAS file.

Text Files

Like files created in the EDT Modes and in the Minitab Mode, files created in the BASIC Mode are text files, which contain only ASCII characters. These are the simplest kind of files, easily transferred from one kind of computer to another, and easily printed by any kind of printer. Each element of the file is a single ASCII character which requires exactly one byte (eight bits) of storage, either in main memory or on disk.

VAX users recognize at least two good reasons for being aware of which files are text files: first, text files are the only files that can be displayed using the TYPE and PRINT commands; and second, they are the only files that can be modified by the EDT editor.

USING THE EDT EDITOR

Using the EDT editor to write or modify BASIC programs is easy. To appreciate this, invoke the EDT editor on the SQUAREROOT.BAS file now, and then use the Keypad Mode to modify it to match the form shown in Fig. 7-11.

```
100      PRINT "Enter X:"
105      INPUT X
110      IF X >= 0 THEN GOTO 120
112      PRINT "X cannot be negative"
116      GOTO 105
120      PRINT SQRT(X)
130      END
```

Fig. 7-11. Using the EDT Keypad Mode to modify a BASIC program.

```
$ BASIC

VAX BASIC V2.4

Ready

OLD SQUAREROOT
Ready

LIST
SQUAREROOT     25-JUL-1986 09:44

100       PRINT "Enter X:"
105       INPUT X
110       IF X >= 0 THEN GOTO 120
112       PRINT "X should not be negative."
116       GOTO 105
120       PRINT SQRT(X)
130       END

Ready

RUN
SQUAREROOT      25-JUL-1986 09:45

Enter X:
? -25
X should not be negative.
Enter X:
? 25
 5
Ready
```

Fig. 7-12. Running the modified program.

Re-enter the BASIC Mode and check your program as shown in Fig. 7-12. Then, run it again to test it out.

It might seem preferable to do all of your work in the EDT editor, and never use the BASIC Mode. To run a BASIC program, however, you must either return to the BASIC Mode, or you must compile and link your program. (See Self-Test Exercise 5.) Obviously, running a new or modified program is easier in the BASIC Mode.

Save your current program, and then create the text-processing program shown in Fig. 7-13. Copy this program exactly as it appears here, including any errors. Now, try to RUN it as in Fig. 7-14.

Syntax Errors

This shows how the BASIC Mode responds to syntax errors. The "1" points to the "L" in "LAST," and the error message says that this is an identifier which is being ignored. The system found where the error is, referring to "line 7." That means that the error is on the seventh buffer line, which is line number 160 in the BASIC program. The problem is that a semicolon (or a comma) is needed after the quotation marks. Make that correction now, and then try to run it again, following Fig. 7-15.

```
Ready

SAVE
Ready

NEW  GREETINGS
Ready

SEQ
100    PRINT "Last name";
110    INPUT LAST$
120    PRINT "First name";
130    INPUT FIRST$
140    PRINT "Hello, "; LAST$; ", "; FIRST$
150    PRINT "Goodbye, "; LEFT$(FIRST$,1);
160    PRINT ". " LAST
170    END
180    <CTRL/Z>
Ready
```

Fig. 7-13. Writing a text-processing program.

```
RUN
GREETINGS    25-JUL-1986 09:51

Error on source line 7, BASIC line 160

        160     PRINT ". " LAST
..........................1
%BASIC-E-DELETE, 1          ignoring user identifier
Ready
```

Fig. 7-14. Running the program. This program has a syntax error on line 160.

```
160     PRINT ". "; LAST
RUN
GREETINGS    25-JUL-1986 10:03

Last name? Gorbachev
First name? Mikhail
Hello, Gorbachev, Mikhail
Goodbye, M.  0
Ready
```

Fig. 7-15. Debugging and rerunning the program.

Logical Errors

Now, the system runs the program without error—at least, without any errors that are detectable by the system. The program still isn't working quite right, however. On the last line, the integer zero is printed, instead of the name "Gorbachev," which is the value of the string variable LAST$. This is an example of a logical error. The system has no way of detecting the error, because the program is correct from its point of view. The problem is that the variable LAST is different from the variable LAST$. Without the trailing dollar sign ($), it is

interpreted as a numerical variable, and as such, it is given the value of zero when the program begins to execute. That value, therefore, is printed right after the value of LEFT$(FIRST$,1), on the last line. All you need to do is insert the dollar sign at the end of LAST on line 170. Because you are still in BASIC Mode, you could do this by retyping line 170. It's probably easier, however, to use the EDT editor to make such a simple change. Fortunately, you can do that without even leaving the BASIC Mode. The VAX allows you to invoke the EDT editor directly from within the BASIC Mode.

In the BASIC Mode, the EDIT command invokes the EDT editor, and from there the EXIT command returns to the BASIC Mode, so you could invoke the editor and then edit line 15. But, if you didn't know that the error was on line 7, you'd have to do a search first. A simpler approach is to use the BASIC line number 160. Single substitutions can be made directly, without leaving the BASIC Mode, by using this modified EDIT command. Do this now, as shown in Fig. 7-16, to fix the program, and then run it again.

```
Ready

EDIT 160/ST/ST$/

160      PRINT ". "; LAST$

Ready

LIST
GREETINGS     25-JUL-1986 10:11

100    PRINT "Last name";
110    INPUT LAST$
120    PRINT "First name";
130    INPUT FIRST$
140    PRINT "Hello, "; LAST$; ", "; FIRST$
150    PRINT "Goodbye, "; LEFT$(FIRST$,1);
160    PRINT ". "; LAST$
170    END

Ready

RUN
GREETINGS     25-JUL-1986 10:03

Last name? Gorbachev
First name? Mikhail
Hello, Gorbachev, Mikhail
Goodbye, M. Gorbachev
Ready
```

Fig. 7-16. Using the EDIT command to invoke the EDT editor from within the BASIC Mode.

The EDIT command in the BASIC Mode works the same way that the SUBSTITUTE command in the EDT Mode works, except that the line number is given first, instead of last.

Now that the program is working correctly, some explanations are in order. In VAX-11 BASIC, the semicolon at the end of the line keeps the cursor (or print

head) on the same line after it executes its PRINT statement. So, in this program, the INPUT statements receive their input on the same line as the prompts provided by the program. Also, in most dialects of BASIC, a dollar sign at the end of the variable identifier declares that variable to be a *string variable*; that is, its value is a character string like 'Gorbachev,' instead of a number like zero.

Stopping a Program

There are times when you will want to stop the execution of a program before it finishes normally. This can always be done by pressing <CTRL/Y>. When a program is run by executing the RUN command from the DCL Mode, this method of aborting is as good as any. If your program is running within the BASIC Mode, however, pressing <CTRL/Y> is usually not the best way to abort it.

Earlier in this chapter, you saw that if you are in any submode of the DCL Mode, the <CTRL/Y> command will interrupt that submode and return you to the DCL Mode. When this happens, the contents of your current buffers will be lost (unless you return immediately to the submode by means of the CONTINUE command). Consequently, when you want to stop the execution of a program from within the BASIC Mode, you should use <CTRL/C> instead of <CTRL/Y>. This will keep you in the BASIC Mode, so that the program in your buffer is not lost. And, as with the <CTRL/Y> command, you can resume the execution of your program by typing CONTINUE.

Interrupting the program with <CTRL/C> causes the loss of some input. In the example in Fig. 7-17, the VAX reads the empty string for the value of LAST$. SAVE the GREETINGS program for use in Self-Test Exercise 5.

```
Ready

RUN
GREETINGS    25-JUL-1986 10:18

Last name? <CTRL/C>
Ready

CONTINUE

First name? Mikhail
Hello, , Mikhail
Goodbye, M.
Ready

RUN
GREETINGS    25-JUL-1986 10:19

Last name? Gorbachev
First name? <CTRL/Y>
 Interrupt

$
```

Fig. 7-17. Interrupting the execution of a BASIC program. The <CTRL/Y> interrupt exits the BASIC Mode and returns to the DCL Mode. The <CTRL/C> interrupt leaves your process in the BASIC Mode.

SELF-TEST EXERCISES

1. Write and run a VAX-11 BASIC program to compute and print the sum of two numbers.
2. Write and run a VAX-11 BASIC program to print the larger of two numbers. This will require the use of the IF statement.
3. In BASIC Mode, type HELP COMMANDS to get a list of all of the commands available in this mode. Then, try some of them, including the DELETE, IDENTIFY, RENAME, RESEQUENCE, RUNNH, SCRATCH, and UNSAVE commands.
4. In BASIC Mode, type HELP to get a list of all of the statements allowable in VAX-11 BASIC. Then, try some of them in programs, including the DATA, MID$, RND, and TIME$ commands.
5. In DCL Mode, compile, link, and run one of your BASIC programs using the DCL commands BASIC, LINK, and RUN. For example, to do this for the program GREETINGS, type:

   ```
   $ BASIC P2
   $ LINK P2
   $ RUN P2
   ```

   Notice that this creates the object module P2.OBJ and the executable image P2.EXE, which are not text files.
6. [For Pascal programmers] In a Pascal program, an error like the one made with LAST$ would be a compile-time error instead of a logical error. Why? From this point of view, which is a better programming language: BASIC or Pascal?

Chapter 8

VAX/VMS Subdirectories and Protection Codes

I think that I shall never see A poem lovely as a tree.

Joyce Kilmer

In this chapter, two features of the VAX/VMS operating system will be investigated which make life easier for the serious user. *Subdirectories* allow you to keep your files nicely organized and easier to handle. *Protection codes* allow you to restrict and permit access to your files in several ways.

CREATING SUBDIRECTORIES

As you accumulate more files in your directory, you will have more difficulty finding things quickly. Moreover, copying and deleting groups of files using wildcards will become more difficult. Fortunately, the VAX provides a very effective way to organize your files in order to overcome these problems. This is done by using subdirectories.

When you first get your account, you have only one directory. If your username is REAGANRON, then your directory is [REAGANRON]. This directory name is part of the filespec for each file in your directory. However, you can, at any time, partition and subpartition this directory into subdirectories. The two commands that allow you to do this are the CREATE/DIRECTORY command and the SET DEFAULT command.

Suppose you have several dozen text files that contain letters to friends and colleagues, and you would like to keep these files separate from all of your others. A simple way to do this is to create a subdirectory, and then move those files into it. Follow the example in Fig. 8-1 to see how this works.

```
$ CREATE TEST.TXT
<CTRL/Z>
$ DIRECTORY TEST.TXT

Directory STS$USER:[REAGANRON]

TEST.TXT;1

Total of 1 file.
$ CREATE/DIRECTORY [.LETTERS]
$ DIRECTORY *.DIR

Directory STS$USER:[REAGANRON]

LETTERS.DIR

Total of 1 file.
$ SHOW DEFAULT
  SYS$USER:[REAGANRON]
$ SET DEFAULT [.LETTERS]
$ SHOW DEFAULT
  SYS$USER:[REAGANRON.LETTERS]
$ DIRECTORY
%DIRECT-W-NOFILES, no files found
$ DIRECTORY [-]TEST.TXT

Directory STS$USER:[REAGANRON]

TEST.TXT;1

Total of 1 file.
$
```

Fig. 8-1. Executing the CREATE, CREATE/DIRECTORY, SHOW DEFAULT and SET DEFAULT commands. This creates a subdirectory and then sets this new directory as the default (current) directory.

The CREATE/DIRECTORY Command

First, the CREATE command is used to create an empty test file named TEST.TXT. The DIRECTORY command verifies that the file is in the default directory [REAGANRON]. Next, the CREATE/DIRECTORY command is used to create the new subdirectory. The name of this subdirectory is LETTERS. Notice that the square brackets with the period, [.], are required syntax for this command.

One effect of the CREATE/DIRECTORY command is the creation of a new file in the current directory. This new file is named LETTERS.DIR. Its existence is seen from the next DIRECTORY command. The file LETTERS.DIR is a *directory file.* Every subdirectory exists by virtue of its directory file, which are not ASCII text file.

The Default Directory

To use the new subdirectory effectively, you must make it the default directory. The SHOW DEFAULT command displays the name of the current default directory. The SET DEFAULT command changes the default directory. Note that, once the default directory has been changed to [REAGAN.RON.LETTERS], no files are listed by the DIRECTORY command. Remember that DCL commands (like DIRECTORY, COPY, DELETE, etc.) that refer to files will search only through the current default directory unless the filespecs include other directories.

The Parent Directory

The last command in Fig. 8-1 is a request to see all files of the form TEST.TXT in the *parent directory.* The parent directory is indicated by the symbol [-]. If the default directory is [REAGANRON.LETTERS], then the parent directory is [REAGANRON]. The DIRECTORY command shows that the TEST.TXT file is still there.

Suppose that TEST.TXT is one of the letters that you want to keep in the LETTERS subdirectory. The steps in Fig. 8-2 show how to move the file down to that subdirectory. The SHOW DEFAULT and DIRECTORY commands (abbreviated SHO DEF and DIR) establish that you are in the LETTERS subdirectory and that it is empty. The RENAME command is used to move the file TEST.TXT from its current [REAGANRON] directory into the default subdirectory [REAGANRON.LETTERS]. The DIRECTORY commands then verify that the file is indeed in [REAGANRON.LETTERS], and not in [REAGANRON].

```
$ SHO DEF
  SYS$USER:[REAGANRON.LETTERS]
$ DIR
%DIRECT-W-NOFILES, no files found
$ RENAME [-]TEST.TXT []*.*
$ DIR

Directory SYS$USER:[REAGANRON.LETTERS]

TEST.TXT;1

Total of 1 file.
$ DIR [-]TEST.TXT
%DIRECT-W-NOFILES, no files found
$
```

Fig. 8-2. Executing the SHOW DEFAULT command and using the RENAME command to move a file from the parent directory to the current directory.

THE TREE DIAGRAM

Subdirectories of subdirectories are as easy to create and use as subdirectories of your original default directory. Your original default directory is called your *root directory.* Together with all its subdirectories, it forms a hierarchy of

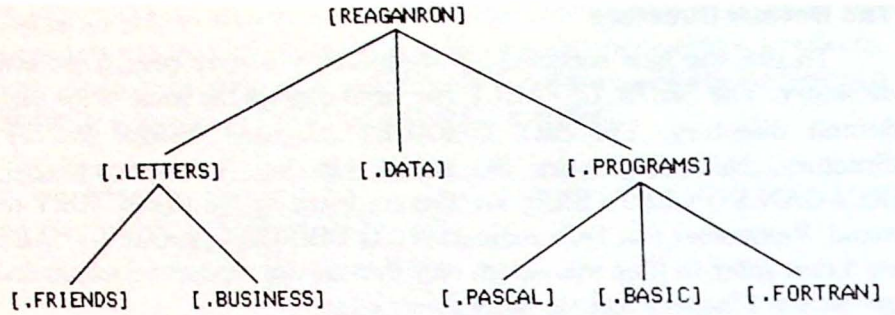

Fig. 8-3. A tree diagram showing a possible directory structure. The root directory is [REAGANRON].

directories which is best represented by a *tree diagram*. Figure 8-3 shows how your subdirectories might be structured after having created a few more.

The tree diagram shows that [REAGANRON] is the root directory and that it is the parent of the subdirectories [.LETTERS], [.DATA], and [.PROGRAMS]. The diagram also shows that [.LETTERS] is the parent of [.FRIENDS] and [.BUSINESS], and that [.PROGRAMS] is the parent of [.PASCAL], [.BASIC], and [.FORTRAN]. Note that every subdirectory is preceded by a period. If a file named PROGRAM4.BAS;3 were in the subdirectory [.BASIC], then its complete filespec (on the current node) would be:

SYS$USER:[REAGANRON.PROGRAMS.BASIC]PROGRAM4.BAS;3

The periods are used to separate subdirectory names from their parents names. If your current default directory were [REAGANRON.LETTERS.BUSINESS], ad you wanted to refer to this program, then you would use:

[-.-.PROGRAMS.BASIC]PROGRAM4.BAS;3

Each hyphen (-) indicates a parent directory, so from [REAGANRON.PROGRAMS.BASIC], [-] indicates [REAGANRON.PROGRAMS], and [-.-] indicates [REAGANRON].

Follow the example in Fig. 8-4 to see how to create the subdirectories [REAGANRON.LETTERS.BUSINESS] and [REAGANRON.DATA].

First, the [.BUSINESS] subdirectory is created (using the abbreviation CR/D for CREATE/DIRECTORY). Then, a copy of the TEST.TXT file is inserted into the [.BUSINESS] subdirectory. To create the "brother" subdirectory [.DATA], you must first have moved up to the parent root directory, which is [REAGANRON]. Then, CR/D is used to create the [.DATA] subdirectory.

At this point, your subdirectory tree should look like Fig. 8-5. This example shows that you have two files in your [.LETTERS] subdirectory: BUSINESS.DIR;1 and TEST.TXT;1. The first file is the directory file for the [.BUSINESS] subdirectory, and the second is the dummy file moved down from the root directory.

```
$ SHO DEF
  SYS$USER:[REAGANRON.LETTERS]
$ CR/D [.BUSINESS]
$ DIR

Directory SYS$USER:[REAGANRON.LETTERS]

BUSINESS.DIR;1      TEST.TXT;1

Total of 2 files.
$ COPY TEST.TXT [.BUSINESS]
$ DIR [.BUSINESS]

Directory SYS$USER:[REAGANRON.LETTERS.BUSINESS]

TEST.TXT;1

Total of 1 file.
$ SET DEF [-]
$ SHO DEF
  SYS$USER:[REAGANRON]
$ CR/D [.DATA]
$ DIR *.DIR

Directory SYS$USER:[REAGANRON]

DATA.DIR;1          LETTERS.DIR;1

Total of 2 files.
```

Fig. 8-4. Creating more subdirectories and COPYing a file down to one of them.

Scope

To illustrate the notion of the *scope* of a command, move to the [.LETTERS] subdirectory and try to delete everything from that point. Follow Fig. 8-6.

The Asterisk

The command DEL *.*;* will delete everything that is deletable in the current subdirectory. In the [.LETTERS] subdirectory, only the TEST.TXT;1

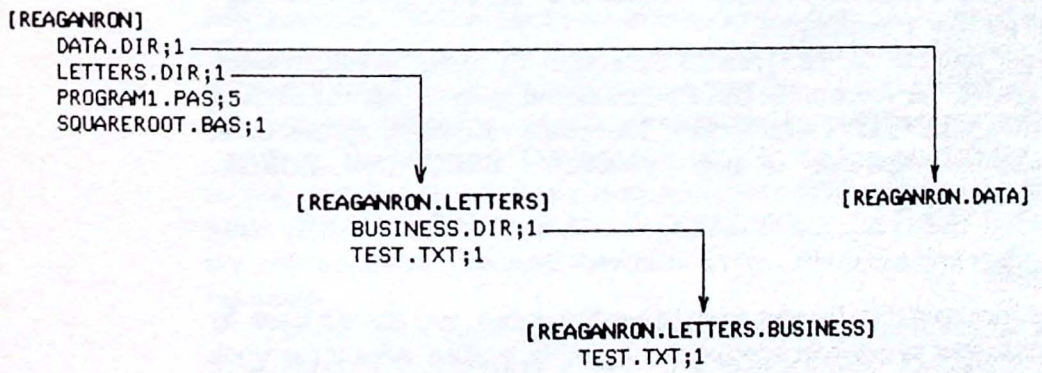

Fig. 8-5. The current structure and contents of the account. This shows three directory files (DATA.DIR, LETTERS.DIR, and BUSINESS.DIR) and how they point to their corresponding subdirectories.

```
$ SET DEF [REAGANRON.LETTERS]
$ DIR

Directory SYS$USER:[REAGANRON.LETTERS]

BUSINESS.DIR;1       TEST.TXT;1

Total of 2 files.

$ DIR [.BUSINESS]

Directory SYS$USER:[REAGANRON.LETTERS.BUSINESS]

TEST.TXT;1

Total of 1 file.

$ DEL *.*;*
%DELETE-W-FILNOTDEL, error deleting STS$USER:[REAGANRON.LETTERS]-
BUSINESS.DIR;1
-RMS-E-PRV, insufficient privilege or file protection violation
$ DIR

Directory SYS$USER:[REAGANRON.LETTERS]

BUSINESS.DIR;1

Total of 1 file.

$ DIR [.BUSINESS]

Directory SYS$USER:[REAGANRON.LETTERS.BUSINESS]

TEST.TXT;1

Total of 1 file.
```

Fig. 8-6. Executing the DELETE command. This shows that the scope of a command is normally limited to the current directory. It also shows that directory files are automatically protected against deletion.

file was deletable. The error message reports that the directory file BUSINESS.DIR;1 could not be deleted because of a "file protection violation." You shall see how to fix this soon.

Finally, note that the TEST.TXT;1 file in the [.BUSINESS] subdirectory was not deleted. The scope of the DELETE command covered only the current [REAGANRON.LETTERS] subdirectory. This scope restriction applies to all DCL commands which refer to files, e.g., COPY, DIRECTORY, PURGE, SEARCH, and TYPE.

The Ellipsis

You have seen that, to refer down to subdirectories, you specify them by name, and to refer up to parent directories, you use the hyphen. Another symbol, which is particularly useful when searching for a file, is the ellipsis (...). This symbol refers to all subdirectories of the given depth. Follow the example in Fig. 8-7 to see how this works.

```
$ SET DEF [REAGANRON]
$ DIR [...]

Directory SYS$USER:[REAGANRON]

DATA.DIR;1          LETTERS.DIR;1       PROGRAM1.PAS;5
SQUAREROOT.BAS;1

Total of 4 files.

Directory SYS$USER:[REAGANRON.LETTERS]

BUSINESS.DIR;1

Total of 1 file.

Directory SYS$USER:[REAGANRON.LETTERS.BUSINESS]

TEST.TXT;1

Total of 1 file.

Grand total of 3 directories, 6 files.
$ DIR [...]*.DIR

Directory SYS$USER:[REAGANRON]

DATA.DIR;1          LETTERS.DIR;1

Total of 2 files.

Directory SYS$USER:[REAGANRON.LETTERS]

BUSINESS.DIR;1

Total of 1 file.

Grand total of 2 directories, 3 files.
$
```

Fig. 8-7. Using the ellipsis (. . .) wildcard to access all subdirectories.

Here, the DIR [...] command lists all files in the current directory and in all subdirectories. Notice that it concludes with a summary, which gives the number of directories mentioned and the total number of files listed. Also note that the command only lists the directories which have files. The [REAGANRON.DATA] subdirectory still exists, but it is not mentioned because there are no files in it.

Next, the DIR [...]*.DIR is used to get a list of all the directory files. This is an easy way to see complete structure of your subdirectory tree. Note, directories [REAGANRON.DATA] and [REAGANRON.LETTERS.BUSINESS] are not mentioned, because they have no files which conform to the *.DIR template.

Finally, note the important distinction between a directory file and the actual subdirectory to which it points. For example, the DIR [...] command lists the directory file LETTERS.DIR;1 first, and then it lists the actual directory [REAGANRON.LETTERS]. Figure 8-5 emphasized this distinction. An analogous situation would be a reference book which lists all the different telephone

books for the telephone company. One of the items listed in the book would be the name of the Manhattan telephone book. That one-line name, however, would not be the same as the actual Manhattan telephone book, which itself, would contain many other names.

FILE PROTECTION

Recall that when you used the DEL *.*;* command, an error message was produced (see Fig. 8-6), which reported that the file BUSINESS.DIR;1 in the directory [REAGANRON.LETTERS] was protected from deletion. The file in question is the directory file for the subdirectory [.BUSINESS]. All directory files are automatically protected from deletion this way when they are created. This can be overcome, however, as shown in Fig. 8-8. After moving to the [.LETTERS] subdirectory, the SET PROTECTION command is given. The SET PROTECTION command requires a *value*. In DCL, a value is something that is assigned to a variable. In the following command,

SET PROTECTION = (OWNER:D) BUSINESS.DIR

the expression (OWNER:D) is the value to be assigned to the PROTECTION variable. DCL requires the equal sign for a value to be assigned to a variable.

The SET PROTECTION command also requires a parameter. In this example, the parameter is the filespec BUSINESS.DIR. The meaning is obvious: you are telling the VAX/VMS operating system to let the owner of the file BUSINESS.DIR be allowed to delete it.

Now, when you attempt to delete the file directory, you no longer get the "file protection violation" message. But, you do get the DIRNOTEMPTY message. A subdirectory cannot be deleted unless it is empty, and the DIR command shows that the [.BUSINESS] subdirectory still has a file in it. Therefore, you must empty the [.BUSINESS] subdirectory by deleting the file [.BUSINESS]TEST.TXT; and then delete the directory file BUSINESS.DIR;. Note

```
$ SET DEF [REAGANRON.LETTERS]
$ SET PROTECTION = (OWNER:D) BUSINESS.DIR
$ DEL BUSINESS.DIR;
%DELETE-W-FILNOTDEL, error deleting SYS$USER:[REAGANRON.LETTERS]-
BUSINESS.DIR;1
-RMS-E-MKD, ACP could not mark file for deletion
-SYSTEM-F-DIRNOTEMPTY, directory file is not empty
$ DIR [.BUSINESS]

Directory SYS$USER:[REAGANRON.LETTERS.BUSINESS]

TEST.TXT;1

Total of 1 file.
$ DEL [.BUSINESS]TEST.TXT;
$ DEL BUSINESS.DIR;
$
```

Fig. 8-8. Executing the SET PROTECTION command. This clears the protection on the directory file BUSINESS.DIR. But it still cannot be deleted until all the files in its subdirectory have been deleted.

that appending a semicolon (;) to the filespec without a version number indicates the latest version.

The User Identification Code

On the VAX, every user is assigned a *user identification code* (UIC) when the user's account is created. Then, whenever the user creates a new file (e.g., with the EDT editor), that UIC is assigned to the new file and the user is designated as the owner of the file. You can see your UIC by using the command qualifier /OWNER with the DIRECTORY command, as in Fig. 8-9. This user's UIC is [STUDENT,REAGANRON].

```
$ DIR/OWNER [REAGANRON]

Directory SYS$USER:[REAGANRON]

PROGRAM1.PAS;5          [STUDENT,REAGANRON]
SQUAREROOT.BAS;1        [STUDENT,REAGANRON]

Total of 2 files.
$ SHOW PROTECTION
  SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD=NO ACCESS
$ DIR/PROTECTION [REAGANRON]

Directory SYS$USER:[REAGANRON]

PROGRAM1.PAS;5          (RWED,RWED,RE,)
SQUAREROOT.BAS;1        (RWED,RWED,RE,)

Total of 2 files.
```

Fig. 8-9. Executing the DIRECTORY command with the /OWNER and /PROTECTION qualifiers.

A UIC has two parts: the *group code* and the *member code*. In the example in Fig. 8-9, the user's group code is STUDENT, and the member code is REAGANRON. Some UICs consist of two numbers instead of two names. For example, a UIC might be [201,037], with group code 201 and member code 037. All users with the same group code form a UIC group. So, not only does every file have a unique owner, but it also belongs to a unique UIC group. In the example in Fig. 8-9, REAGANRON is the name of the owner of the file PROGRAM1.PAS;5, and STUDENT is the name of the group for that file. So, the user with UIC [STUDENT,GORBACHEV] is in the file's group, but the user with UIC [FACULTY,WIRTHNIKL] is not.

User Categories

Actually, for each file, four user categories are defined. The smallest category is called the SYSTEM category. It contains only those users who have system privileges, namely, the system manager and, perhaps, a few others. The second category, called the OWNER category, contains only the owner of a particular file. Next, the GROUP category contains all users in a file's UIC group.

Finally, the WORLD category includes all users. These four user categories are used to define the file's *protection code.*

Every file has a four-part protection code which the system uses to determine what type of access is to be given to each category of users. When the file is created, it is automatically given the *default* protection code assigned to its owner. Give the SHOW PROTECTION command to see what your default protection code is. The example in Fig. 8-9 shows that this user's default protection code is:

SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD=NO ACCESS

This means that both the owner and all users in the system category may Read, Write, Execute, or Delete any file with this code. A member of the file's group may only Read or Execute it, and all others are given no access. Every file created by this user will be given this protection code.

Types of Access

To see the protection code assigned to the files themselves, use the DIRECTORY/PROTECTION command. Figure 8-9 shows that the two files in the [REAGANRON] directory have the default protection code described above. Notice how the code is abbreviated. The four parts of the code represent the four user categories SYSTEM, OWNER, GROUP, and WORLD, in that order. Each part might have any (or none) of the four letters R, W, E, or D, which stand for Read access, Write access, Execute access, and Delete access. In protection codes, the access types must be abbreviated to one letter.

If a file's protection code allows Read access to a user, that user will be able to use commands like COPY, EDIT, PRINT, READ, and TYPE on the file. (Of course, the EDIT, PRINT and TYPE commands are useless except on ASCII text files.) If the protection code allows Write access to a user, that user will be able to use commands like WRITE on the file; the file can be changed by the user. If a file's protection code allows Execute access to a user, that user will be able to use the RUN command on the file. (Of course, the file must contain an executable image to be RUN.) If a file's protection code allows Delete Access to a user, that user will be able to DELETE and PURGE the file. Figure 8-10 summarizes these facts.

Thus, Fig. 8-9 shows that the owner, and everyone with system privileges, is able to TYPE, EDIT, and DELETE the file PROGRAM1.PAS;5. Everyone in

```
Access Type   Allowable Commands

READ          COPY, EDIT, PRINT, READ, TYPE, WRITE
WRITE         WRITE
EXECUTE       RUN
DELETE        DELETE, PURGE
```

Fig. 8-10. The four access types and the DCL commands which they allow.

the owner's group can TYPE and EDIT the file, but not DELETE it, and all others have no access at all to the file. A default protection code is given to each file that a user creates. The SHOW PROTECTION command displays the user's current default protection code. The SET PROTECTION/DEFAULT command allows the user to change the default protection. A changed default protection will remain in effect until it is changed again, or until the user logs out of the system.

Follow the example in Fig. 8-11 to change your default protection. Access for both the SYSTEM and OWNER category is reset to RED, and access for the GROUP category is set just to R. Also, WORLD access is set to E, because in this example, you are willing to let anyone RUN our programs.

```
$ SET PROTECTION=(S:RED,O:RED,G:R,W:E)/DEFAULT
$ SHOW PROTECTION
  SYSTEM=RED, OWNER=RED, GROUP=R, WORLD=E
$ CREATE TEMP.TMP
<CTRL/Z> Exit
$ DIR/PROTECTION [REAGANRON]

Directory SYS$USER:[REAGANRON]

PROGRAM1.PAS;5          (RWED,RWED,RE,)
SQUAREROOT.BAS;1        (RWED,RWED,RE,)
TEMP.TMP;1              (RED,RED,R,E)

Total of 3 files.
```

Fig. 8-11. Executing the SET PROTECTION command with the /DEFAULT qualifier to change the default protection.

Now, when you create a new file, either with the CREATE command or with the EDT editor, the file is automatically given this new protection code.

Suppose you decide that you don't want anyone in your group even to be able to see your PROGRAM1.PAS;5 file. You can change the protection on any file or group of files by executing the SET PROTECTION command with the file(s) listed as parameter(s) to the command. See Fig. 8-12.

```
$ SET PROT=(G) PROGRAM1.PAS;5
$ DIR/PROT PROGRAM1.PAS;5

Directory SYS$USER:[REAGANRON]

PROGRAM1.PAS;5          (RWED,RWED,,)

Total of 1 file.
$ SET PROT=(O:RE,W:R) PROGRAM%.PAS;*
$ DIR/PROT PROGRAM1.PAS;5

Directory SYS$USER:[REAGANRON]

PROGRAM1.PAS;5          (RWED,RE,,R)

Total of 1 file.
$
```

Fig. 8-12. Executing the SET PROTECTION command.

The GROUP user category is indicated by the G in the protection code. Notice that no colon is used. This means that the new access code for the GROUP category is to be NO ACCESS. The DIRECTORY/PROTECTION command (abbreviated DIR/PROT) reveals that the protection code for the PROGRAM1.PAS;5 file has been changed. The four categories SYSTEM, OWNER, GROUP, WORLD are always referenced in that order, so the third category is the GROUP category. There are now no letters between the second and third commas in the code for the PROGRAM1.PAS;5 file. That shows that, now, those users in the file's UIC group will not be able to access the file in any way.

SYNTAX FOR THE SET PROTECTION COMMAND

The SET PROTECTION command has the following rather specific syntax requirements:

- Each access type must be abbreviated by its first letter: W for write, R for read, E for execute, and D for delete.
- A user category name may be truncated to any number of letters.
- If a user category is to be denied all access, then that category is listed (or abbreviated) without a colon.
- If a user category is omitted from the specification, then that category retains its previous or default protection code.
- If more than one user category is specified, then the categories must be separated by commas and enclosed in parentheses.

Thus, for example, the command:

```
$ SET PROT=(O:RE,W:R) PROGRAM%.PAS;*
```

would prevent the owner from deleting any file whose filespec fits the wildcard template specified. It would also allow all users to read it. Because the SYSTEM and GROUP categories are not mentioned, their access would not be changed by this command.

Protection codes for directory files require special attention, because the protection code for the directory file of a subdirectory affects the access to all files in that subdirectory. For this reason, it is best, in general, not to allow any access to your directory files.

If REAGANRON executes the command

```
$ RENAME [GORBACHEV.KGB]NAMES.DAT [REAGANRON.OBLIVION]STUFF.DAT
```

and if the [GORBACHEV]KGB.DIR file allows READ access to WORLD, then even if the NAMES.DAT file protection code is NO ACCESS to all users, the command will execute, and GORBACHEV will not be able to find his file. Of course, the file will still be protected from REAGANRON's perusal, but as far as GORBACHEV knows, the file is gone!

To prevent someone else from renaming your files into oblivion, you should ensure that your directory files do not allow Read access to the WORLD category. To do that, follow Fig. 8-13. The first command resets the default directory to the system root directory. This is grandfather directory [000000], which is the parent directory of all user account directories. From here, you can see what protection code the system default has attached to your account. In this example, that protection code is (RWED,RWE,,R). This prevents a user (without system privileges) from deleting his or her own account. But, it also gives every user on the system Read access to his or her account. So, GORBACHEV can give the command DIRECTORY [REAGANRON] and see a listing of all files. He can also rename REAGANRON's files into oblivion, as described above. To avoid this, REAGANRON resets the protection code on his main directory file to (RWED,RWE,,), thereby eliminating the Read access to the WORLD.

```
$ SET DEF [REAGANRON.-]
$ DIR/PROT REAGANRON.DIR

Directory SYS$USER:[000000]

REAGANRON.DIR;1       (RWED,RWE,,R)

Total of 1 file.
$ SET PROT=(W) REAGANRON.DIR
$ DIR/PROT REAGANRON.DIR

Directory SYS$USER:[000000]

REAGANRON.DIR;1       (RWED,RWE,,)

Total of 1 file.
$
```

Fig. 8-13. Changing the protection on your root directory to prevent other users from reading your directories.

SELF-TEST EXERCISES

1. Make your [.DATA] subdirectory your current directory. Then carry out the following steps:
   a. Create subdirectories called OLD and NEW.
   b. Make your [.DATA.NEW] subdirectory your current directory, and create subdirectories [.STUFF] and [.MISC].
   c. Copy some files from your [.DATA] directory into your [.DATA.NEW.STUFF] directory.
2. Change your default protection to (RWED,RED,R,E). Then, create some files and check that they have inherited this default protection code.
3. Check the protection codes on all of your directory files. If any of them allow Read Access to the WORLD, change the access so that they don't.

4. Give the HELP CREATE /DIRECTORY command to see what the HELP files can tell you. Check the Examples subtopic, Parameters subtopic, the Command_Qualifiers subtopic, the /OWNER_UIC subtopic, and the /PROTECTION subtopic.
5. Give the HELP SET PROTECTION command to see what the HELP files can tell you. Check the Examples subtopic, the /DEFAULT subtopic, the /DEFAULT Parameters subtopic, and the /DEFAULT Examples subtopic.

Chapter 9

Command Procedures

The arithmetical machine produces effects which approach nearer to thought than all the actions of animals. But it does nothing which would enable us to attribute will to it.

Blaise Pascal

A *command procedure* is a sequence of DCL commands kept in a single file. The file type must be COM. The commands in the file will be executed automatically when the command procedure is invoked. For example, if TEST.COM is the filespec for a command procedure, then the DCL command @TEST will invoke the command procedure. A command procedure can be regarded as a primitive program of DCL commands. In other computers' operating systems, like MS-DOS, command procedures are called *batch files.* In UNIX (see Chapter 10), they are called *shell procedures.*

As a first example, a command procedure will be created to execute a few of the SHOW commands. Follow Fig. 9-1.

The name of this command procedure is SHOW_STUFF. (Recall that the underscore can be used in VAX/VMS identifiers. It is often used to indicate a blank character, which is not allowed.) When this command procedure is invoked, the VAX will read its lines, just as though they were being typed at the keyboard.

Each line in the command procedure begins with the dollar sign, and con-

```
$ CREATE SHOW_STUFF.COM
$ SHOW DEFAULT
$ SHOW QUOTA
$ SHOW USERS
$ EXIT
<CTRL/Z>
$ TYPE SHOW_STUFF.COM
$ SHOW DEFAULT
$ SHOW QUOTA
$ SHOW USERS
$ EXIT
$
```

Fig. 9-1. Creating a simple command procedure to execute several SHOW commands.

tains a DCL command. The last line contains the DCL command EXIT, which is used only in command procedures. It terminates the execution of the command procedure and returns control to the previous environment, even if it is not the last line in the command procedure.

The SHOW_STUFF.COM file is like any other text file in your directory. You may TYPE it, COPY it, DELETE it, etc. And, because its file type is COM, you may also apply the @ command to it. This "commercial at" character (@) stands for "Execute Procedure." When it is typed with the filename of a command procedure, it invokes the command procedure, causing its commands to execute. As Fig. 9-2 shows, the commands execute automatically, one after the other, just as though they had been typed interactively at the keyboard.

USES FOR COMMAND PROCEDURES

Command procedures are often used to execute frequently used blocks of commands. For example, Pascal programmers frequently have to change their source code to correct mistakes in their programs. This requires invoking the EDT editor, making the changes, and then compiling and linking the program again. (See Chapter 5.) These steps are combined into a single command procedure in Fig. 9-3, called FIX.COM. You might want to use the EDT editor instead of the CREATE command to write this file. This command procedure will be used to facilitate debugging a Pascal program called TEST.PAS. It illustrates the use of comments, the SET VERIFY command, and the ASSIGN/USER_MODE command.

```
$ @SHOW_STUFF
  SYS$USER:[REAGANRON]
  User [STUDENT,REAGANRON] has 77 blocks used, 323 available,
  of 400 authorized and permitted overdraft of 100 blocks on SYS$USER
          VAX/VMS Interactive Users
            2-AUG-1986 08:31:47.13
     Total number of interactive users = 3

 Username      Process Name       PID      Terminal
 GORBACHEV     RED_RIDING_HOOD 00000092   TXB7:
 REAGANRON     REAGANRON       0000009E   TXB1:
 THATCHERM     PM              0000008F   TTA0:
$
```

Fig. 9-2. Executing the command procedure.

```
$ CREATE FIX.COM
$ ! Command Procedure FIX.
$ ! This procedure allows interactive editing of the Pascal
program specified.
$ ! It then compiles and links the program and removes unnecessary files.
$ ! If any error occurs, the procedure exits.
$ SET VERIFY
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ EDIT TEST.PAS
$ PASCAL TEST
$ LINK TEST
$ PURGE TEST.PAS
$ DELETE TEST.OBJ;*
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ RUN TEST
$ SET NOVERIFY
$ EXIT
<CTRL/Z>
$
```

Fig. 9-3. Creating a command procedure to debug a Pascal program. This command procedure begins with four lines of comments to describe what it does. It also contains the SET VERIFY and SET NOVERIFY commands which will cause the other commands in the command procedure to be displayed as they are executed. It also contains the ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT: command to allow interactive input.

Comments

Like all computer programs, command procedures should include comments to explain to anyone who TYPEs the file what the procedure does. The exclamation point is used to indicate comments in command procedures. Everything between the exclamation point and the end of the line is ignored by the Command Interpreter.

The SET VERIFY Command

The SET VERIFY command will cause the VAX to display the DCL commands themselves in the command procedure as they are executed. The SET NOVERIFY command inhibits the display. The default is SET NOVERIFY. With a large command procedure, it is useful to SET VERIFY so that you can find errors on the procedure as they occur. After the procedure is thoroughly debugged, you will probably want to remove these two commands.

User Interaction

During the normal execution of a command procedure, the user is unable to interact with the system; control is maintained in the command procedure. Such activities as editing and running programs, however, normally require interaction between the user and the system. The ASSIGN/USER_MODE command allows this interaction during the execution of a command procedure. In general, the ASSIGN command is used to assign a logical name to another logical name. (See the last section of this chapter for further information on logical names.) The logical name SYS$COMMAND refers to commands given interactively at the keyboard. The logical name SYS$INPUT refers to the input received by the VAX for the execution of a command. So, "ASSIGN SYS$COMMAND:

SYS$INPUT:" in this case means that the input for the command should come from the keyboard. (The colons are necessary.) Finally, the qualifier /USER_MODE means that this reassignment, which allows interactive input, is to apply only during the next command listed in the command procedure. Thus, the EDIT command and the RUN command will read their input from the keyboard.

Now, create a dummy Pascal program, as shown in Fig. 9-4, and then apply FIX to it. Notice that the ASSIGN and EDIT commands are displayed as they are executed. When the system pauses to let you edit the PASCAL program, add a write statement, as shown in Fig. 9-4. Then, EXIT the editor to allow the command procedure to continue executing.

```
$ CREATE TEST.PAS
$ program P; begin end.
$ <CTRL/Z>
$ @FIX
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ EDIT TEST.PAS
    1          program P; begin end.
*S/n e/n write('Hello.') e/
    1          program P; begin write('Hello.') end.
1 substitution
*EX
SYS$USER:[REAGANRON]TEST.PAS;2 1 line

$ PASCAL TEST
00001      0  0 program P; begin write('Hello.') end.
                                    1
%PASCAL-E-OUTNOTDECL, (1) OUTPUT not declared in heading
%PASCAL-E-ENDDIAGS, PASCAL completed with 1 diagnostic
$
```

Fig. 9-4. Testing the FIX.COM command procedure. This test first creates a minimal (one line) Pascal program in the file TEST.PAS and then executes the command procedure to debug it.

The next command shown in Fig. 9-4 is the PASCAL command, which compiles the edited program. If you followed this example as shown, however, the compiler will report the OUTNOTDECL error, saying that the program heading does not declare output. This error interrupts the command procedure, causing it to abort before it gets to the next command (the LINK command).

Aborting a Command Procedure

Invoke the command procedure FIX again to fix the program, as shown in Fig. 9-5. This time, the three lines of comments and the SET VERIFY command are displayed. That is because the SET NOVERIFY command never executed in the last call to FIX; the command procedure terminated prematurely, leaving the system in the SET VERIFY state. This points out one of the problems with allowing a command procedure to abort.

The VAX has a simple way to prevent a command procedure from aborting as in the previous example. The solution is to include an ON ERROR THEN

```
$ @FIX
$ ! Command Procedure FIX.
$ ! This procedure allows interactive editing of the Pascal
program specified.
$ ! It then compiles, links and runs the program and removes
unnecessary files.
$ SET VERIFY
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ EDIT TEST.PAS
    1           program P; begin write('Hello.') end.
*S/;/(output);/
    1           program P(output); begin write('Hello.') end.
1 substitution
*EX
$ PASCAL TEST
$ LINK TEST
$ PURGE TEST.PAS
$ DELETE TEST.OBJ;
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ RUN TEST
Hello.
$ SET NOVERIFY
$
```

Fig. 9-5. Using the command procedure to debug the program.

GOTO command at the beginning of the command procedure, which will transfer control to the end of the procedure in case of an error. Follow the example in Fig. 9-6.

```
$ ED FIX.COM
    1       $ ! Command Procedure FIX.
*I 4;$ON ERROR THEN GOTO FIN
    4       $ SET VERIFY
*I 13;$ FIN:
   13       $ SET NOVERIFY
*EX
SYS$USER:[REAGANRON]FIX.COM;3 16 lines

$ @FIX
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ EDIT TEST.PAS
    1           program P; begin write('Hello.') end.
*S/;/=/
    1           program P(output)= begin write('Hello.') end.
1 substitution
*EX
SYS$USER:[REAGANRON]TEST.PAS;5 1 line

$ PASCAL TEST
00001      0  0 program' P(output)= begin write('Hello.') end.
                                   1
%PASCEL-E-SYNSEMI, (1) Syntax: ";" expected
%PASCAL-E-ENDDIAGS, PASCAL completed with 1 diagnostic
$ FIN:
$ SET NOVERIFY
$
```

Fig. 9-6. Using the ON ERROR THEN GOTO command to make the command procedure more robust.

Responding to Errors

This time, a syntax error was intentionally included in the program to see how the command procedure responds when the Pascal compiler finds an error. The response is to GOTO the label "FIN:". A *label* in a command procedure is any identifier with a colon suffix. It may have up to 255 characters, may not include any imbedded blanks, and may not be any DCL command or abbreviation. In this case, the exit label "FIN:" was put just before the SET NOVERIFY command, so that it would still be executed even if an error occurred.

The ON...THEN Command

The ON . . . THEN command is used in command procedures to cause an immediate action whenever a given condition occurs. It is similar to an if... then statement in Pascal or BASIC, except that it acts as though the statement precedes every line. In other words, once the ON...THEN statement has executed, its control remains in effect for the duration of the procedure; it will immediately interrupt the execution of any statement thereafter if the specified condition becomes true. There are four versions of the ON... THEN command. They are:

- ON CONTROL_Y THEN
- ON ERROR THEN
- ON SEVERE_ERROR THEN
- ON WARNING THEN

The first version will cause an interrupt when the user types <CTRL/Y> during the execution of the command procedure. The other three cause an interrupt when an error of the specified severity level occurs. For example, the command ON WARNING THEN CONTINUE would allow the procedure to continue executing if an error of that severity level occurred. (Recall that the error severity levels are: I for Information, W for Warning, E for Error, and F for Fatal (Severe) Error. Normally, execution will be halted if the severity level is E or F.)

In this example, the FIX command procedure works better. It's too specialized, however, because it only works on the TEST.PAS program. Fortunately, there is an easy way to generalize the procedure so that it will work on any Pascal program. Just replace every reference to the filename TEST with the *parameter* value 'P1', as in Fig. 9-7.

The Parameter

A parameter in a command procedure is a variable whose value is assigned when the procedure is invoked. The VAX requires that a command procedure's parameters be named P1, P2, P3, etc. The procedure in Fig. 9-7 uses the one parameter P1. When it is invoked by @FIX TEST, P1 is given the string TEST for its value. Then, everywhere that 'P1' appears in the command procedure, the command interpreter substitutes the value TEST before executing the command. For example, if you wanted to run the procedure on a Pascal program

```
$ ED FIX.COM
    1        $ ! Command Procedure FIX.
*S/TEST/'P1'/W
    7        $ EDIT 'P1'.PAS
    8        $ PASCAL 'P1'
    9        $ LINK 'P1'
   10        $ PURGE 'P1'.PAS
   11        $ DELETE 'P1'.OBJ;
   13        $ RUN 'P1'
    4        $ SET VERIFY
6 substitutions
*EX
SYS$USER:[REAGANRON]FIC.COM;5 16 lines

$ @FIX TEXT
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ EDIT TEST.PAS
    1            program P(output)= begin write('Hello.') end.
*S/=/;/
    1            program P(output); begin write('Hello.') end.
1 substitution
*EX
SYYS$USER:[REAGANRON]TEST.PAS;11 1 line

$ PASCAL TEST
$ LINK TEST
$ PURGE TEST.PAS
$ DELETE TEST.OBJ;
$ ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$ RUN TEST
Hello.
$ SET NOVERIFY
$
```

Fig. 9-7. Using a parameter to make the command procedure more flexible. The variable 'P1' will be replaced by the string typed when the command procedure is executed. When @FIX TEST is typed, PASCAL TEST is executed in place of PASCAL 'P1' for example.

named PRIMES.PAS, you would just type @FIX PRIMES, and every occurrence of 'P1' would be replaced by PRIMES. Similarly, the command PURGE 'P1'.PAS would be read by the command interpreter as PURGE PRIMES.PAS.

THE LOGIN PROCEDURE

When you log in on the VAX, the VMS operating system searches through your main directory for a command procedure named LOGIN.COM. If this procedure is found, it is invoked immediately. In other words, all DCL commands in the procedure LOGIN.COM will be executed automatically, as soon as you log in. This is called your *login procedure*, and it can be very useful. The example in Fig. 9-8 suggests some commands which you might want to put in your login procedure. The SET NOON command (pronounced "set no on") ensures that, even if an error occurs during the execution of a command in the procedure, the command procedure will continue, thereby at least trying to execute every command in the procedure.

The SET PROCESS/NAME command simply gives a new name (in this case, one of Mr. Reagan's favorites) to the current process. This new name will

```
$ CREATE LOGIN.COM
$ SET NOON
$ SET PROCESS/NAME="THE_GIPPER"
$ SET PROTECTION=(GROUP:R,WORLD)/DEFAULT
$ ASSIGN SYS$USER:[REAGANRON.LETTERS.BUSINESS.OLD] OBL
$ ASSIGN SYS$USER:[REAGANRON.PROGRAMS.BASIC.1985] 85BP
$ SET TERMINAL/NONUMERIC
$ DEFINE/KEY/NOLOG PF1 "SET DEFAULT [."
$ DEFINE/KEY/TERMINATE/NOLOG PF2 "SHOW DEFAULT"
$ DEFINE/KEY/TERMINATE/NOLOG 0 "SHOW QUOTA"
$ DEFINE/KEY/TERMINATE/NOLOG ENTER "SHOW USERS"
$ C*OMPILE :== @[REAGANRON]FIX
$ HO*ME :== SET DEFALUT SYS$USER:[REAGANRON]
$ SIL*LY :== SET PROMPT = "NOW WHAT? "
$ V*AX780 :== SET HOST VAX780
$ W132 :== SET TERMINAL/WIDTH=132
$ W80 :== SET TERMINAL/WIDTH=80
$ SHOW KEY/ALL
$ SHOW QUOTA
$ SHOW USERS
$ SET ON
$ EXIT
<CTRL/Z>
$
```

Fig. 9-8. Creating a LOGIN file. This illustrates the SET NOON, SET PROCESS, SET TERMINAL, DEFINE KEY and SHOW KEY commands. Besides defining certain keys, it also defines the logical symbols COMPILE, HOME, SILLY, VAX785, W132, and W80.

be used, for example, in identifying the user when some executes the SHOW USERS command or the SHOW SYSTEM command.

The ASSIGN commands allow you to use specified abbreviations. For example, if you are in your [REAGANRON.DATA] subdirectory, instead of typing PURGE [-.LETTERS.BUSINESS.OLD]*.TMP, you could just type PURGE OBL*.TMP.

The SET TERMINAL/NONUMERIC command allows you to use the keys on the numeric keypad to execute commands. Ordinarily, these keys duplicate the numeric keys on the standard keyboard. They were provided, originally, to facilitate rapid numeric entry by being arranged like a calculator keypad.

The DEFINE/KEY command allows you to tailor your keyboard to your convenience. If you add the /TERMINATE qualifier, then the command will execute as soon as you press the defined key; otherwise, you must press the key and the <RET> for the command to execute. The /NOLOG qualifier prevents the display of a message announcing that the key has been defined.

String Assignment Statements

Following the key definitions is a series of *string assignment* statements. Each of these assigns the string on the right to the symbol on the left. For example, W132 was entered, so the command SET TERMINAL/WIDTH=132 was executed. (This command clears the screen and then compresses it to 132 columns.) The asterisk used in some of these assignments is not part of the symbol being defined. Instead, it indicates the minimal abbreviation of the symbol. Thus, any of the following would reset the default directory to [REAGANRON]: HO, HOM, HOME.

Another kind of command procedure is one which performs repetitive tasks. Suppose that you have eight files, named P1.PAS, P2.PAS, P3.PAS, through P8.PAS in your current directory, and you want to rename them KM1.PAS, KM2.PAS, KM3.PAS, etc., and save them in a subdirectory named [.OLD]. The command procedure shown in Fig. 9-9 will do all of this for you. The variable N is used to count the number of iterations of the loop. Its value is used to identify and rename each file. In a command procedure, the value of a variable is indicated by enclosing it in apostrophes. Therefore, if PN.PAS were used instead of P'N'.PAS, the system would look for a file whose filename was PN instead of P1, P2, P3, etc.

```
$ CREATE RE.COM
$ N = 0
$ LOOP:
$   N = N + 1
$   RENAME P'N'.PAS [.OLD]KM'N'.PAS
$   IF N .LT. 8 THEN GOTO LOOP
$ EXIT
<CTRL/Z>
```

Fig. 9-9. Creating a command procedure with repetition.

The symbol ".LT." stands for the comparison operator "is less than." The comparison operators available for use with numeric variables are summarized in Fig. 9-10.

Abbreviation	Comparison Operator
.EQ.	equal to
.GE.	greater than or equal to
.GT.	greater than
.LE.	less than or equal to
.LT.	less than
.NE.	not equal to

Fig. 9-10. The six comparison operators for command procedures.

Producing Output on the Screen

There are several ways to produce output on the screen from a command procedure. Probably the simplest approach is to use the WRITE SYS$OUTPUT command. In VAX/VMS, the name SYS$OUTPUT is the logical name for the place where output is sent. Under most circumstances,

SYS$OUTPUT means your terminal screen. Consequently, WRITE SYS$OUTPUT means to write whatever is specified on your screen. The command procedure shown in Fig. 9-11 illustrates the WRITE SYS$OUTPUT command.

```
$ CREATE R.COM
$ WRITE SYS$OUTPUT "This command procedure illustrates ", -
                   "loops and the WRITE SYS$OUTPUT command."
$ N = 0
$ LOOP:
$   N = N + 1
$   WRITE SYS$OUTPUT "N = ", N
$   IF N .LT. P1 THEN GOTO LOOP
$ WRITE SYS$OUTPUT "All I ca", "n say is ""Goodbye""."
$ EXIT
<CTRL/Z> Exit
$ @R 5
This command procedure illustrates loops and the WRITE SYS$OUTPUT command.
N = 1
N = 2
N = 3
N = 4
N = 5
All I can say is "Goodbye".
$ @R
This command procedure illustrates loops and the WRITE SYS$OUTPUT command.
N = 1
All I can say is "Goodbye".
$
```

Fig. 9-11. Writing to the screen.

When the procedure runs, the quoted strings that follow the WRITE SYS$OUTPUT command are written on the screen. Any symbols listed without quotation marks are interpreted as variables, and their values are printed. This example shows that objects in the *write list* are separated by commas and a pair of adjacent quotation marks are printed as a single quotation mark. This example also illustrates the DCL *continuation symbol*, which is the hyphen. When a command line ends with this character, the command interpreter goes to the next line to continue the command. Note that no dollar sign is put on the continued line. Finally, notice what happens when no value is given to the parameter P1 — it is automatically initialized to zero. This stops the loop after its first iteration.

Interactive Input

Command procedures can have interactive input much like Pascal and BASIC programs. The DCL command which allows this is the INQUIRE command. The example in Fig. 9-12 illustrates its use.

The purpose of the INQUIRE command is to get a value for a variable interactively, while the procedure is running. The variable is specified immediately after the command word. Following that, a string in quotes can be specified;

```
$ CREATE R1.COM
$ WRITE SYS$OUTPUT "This command procedure illustrates the INQUIRE
command."
$ N = 0
$ IF P1 .NES. "" THEN GOTO LOOP
$ INQUIRE P1 "Enter bound on N"
$ LOOP:
$   N = N + 1
$   WRITE SYS$OUTPUT "N = ", N
$   IF N .LT. P1 THEN GOTO LOOP
$ INQUIRE NAME "What is your name? "
$ WRITE SYS$OUTPUT "All I can say is ""Goodbye, ", NAME, """"
$ EXIT
<CTRL/Z> Exit
$ @R 2
This command procedure illustrates the INQUIRE command.
N = 1
N = 2
What is your name? : JASON
All I can say is "Goodbye, JASON"
$ @R
This command procedure illustrates the INQUIRE command.
Enter bound on N: 3
N = 1
N = 2
N = 3
What is your name? : MIKHAIL
All I can say is "Goodbye, MIKHAIL"
$
```

Fig. 9-12. Using the INQUIRE command to allow interactive input.

if it is, then that string is used as a prompt for the input. For example, the prompt for the input for the variable NAME is "What is your name? " Notice that the system automatically attaches a colon and a blank after the prompt.

This example also illustrates the use of comparison operators for string variables. Figure 9-10 listed the six comparison operators for numeric variables. The corresponding six operators for string variables are the same, except that an "S" (for "string") is attached to each. Thus, the conditional:

IF P1 .NES. " "

tests whether the string variable P is not equal to the string constant " " (which is the empty string). The purpose here is to see if any value was given to the parameter P1 when the procedure was invoked. If no value was given, then as a string variable, P1 is automatically initialized to be the empty string. In this example, if a value were given to P1 when the procedure is invoked, then the condition (not equal to the empty string) would be true, and the GOTO statement would execute, thereby jumping over the first INQUIRE statement.

SELF-TEST EXERCISES FOR CHAPTER 9

1. Write a command procedure which PURGEs a given file and then resets its version number to 1.
2. Write a command procedure which increments the version number of all versions of a given file by one.

3. Create a subdirectory named COMPROCS. Then, move all of your command procedures into it, and modify your LOGIN.COM procedure so that it still accesses the procedures as it did before.
4. Write a command procedure which asks for the user's name and then writes it (on the screen).
5. Write a command procedure which asks for the user's gender. Then, if the gender is female, it asks which form of address she prefers: MISS, MRS, or MS. Next, it asks for the user's last name, and, finally, prints it with the appropriate form: MISS, MR, MS, or MRS.
6. Rewrite the command procedure shown in Fig. 9-12 so that it does the same things, but uses the .EQS. comparison operator instead of .NES. (HINT: Remove the GOTO.)
7. Give the HELP LEXICALS F$EXTRACT command to learn about the *lexical function* F$EXTRACT. A lexical function is a function used in a command procedure, similar to functions used in Pascal and BASIC programs. All DCL lexical function begin with the two characters F$ and end with parentheses. The F$EXTRACT function is used to extract a substring from a specified string. For example, if the value of the string NAME is "JOE SMITH," then the value of the STRING F$EXTRACT(0,3,NAME) is "JOE." Use the F$EXTRACT function in a program which asks for the user's full name, first name first, and then prints the full name, last name first.

Chapter 10

Transition to UNIX

The lord who is the oracle at Delphi neither reveals nor hides, but gives tokens.

Heraclitus

An *operating system* is a collection of programs that work together to control the operation of a computer. One of these programs, sometimes called the *kernel*, is kept in the main memory of the computer and is always running. A copy of the operating system is kept on disk and is loaded into main memory whenever the computer is "powered up."

Every computer must have an operating system. The manufacturer usually supplies one with the computer. Such an operating system is said to be native, because it was designed specifically for that computer. The native operating system for DEC's family of VAX computers is called VMS, an acronym for "Virtual Memory System."

All programs and other software systems that run on a computer must *run under* the computer's operating system. This means that all such software must conform to the specifications of the given operating system. Consequently, computers which use different operating systems require different versions of the same software. The version of Minitab written for DEC's VAX/VMS operating system, for example, is different from the version written for IBM's MVS operating system.

THE UNIX OPERATING SYSTEM

The problem of incompatible operating systems was solved in the early 1970s by D. M. Ritchie and K. Thompson at Bell Laboratories. They wrote a very simple operating system, called UNIX, for DEC's family of PDP computers. In 1974, the system was licensed to universities for educational purposes. It became available for commercial use a few years later. Because the system is so simple, versions soon became available for other types of computers. Now, it is possible to buy a UNIX operating system for just about any computer in the world.

The first nine chapters of this book are tutorials on various modes of the VAX/VMS operating system. This chapter introduces the UNIX operating system and its command language. Chapter 11 introduces the UNIX text editor.

The Native Mode UNIX

There are two kinds of UNIX systems available for VAX computers. One consists of the *native mode* UNIX systems, which run as the controlling operating system of the computer. DEC's ULTRIX is the native mode UNIX system provided by DEC for its VAX computers. Another popular native mode UNIX system for VAX computers is the MORE/bsd system, marketed by Mt Xinu.

The UNIX Emulator

The other kind of UNIX system available for VAX computers is the UNIX *emulator*. This is a software system which is said to run on top of the VAX/VMS system. In other words, VMS is actually controlling the VAX, but to the user, it looks like UNIX. UNIX emulators have the advantage of being compatible with VMS (for example, they use the same file organization and specifications), but they tend to run more slowly, because their actions must be translated into VMS. The UNIX emulator provided by DEC is called DEC/Shell. Another popular UNIX emulator for VAX computers is the EUNICE system, marketed by The Wollongong Group.

Although the work in Chapters 10 and 11 will be based upon the UNIX emulator EUNICE, any other version of UNIX available on the VAX should respond the same way to the steps in these two chapters. Exceptions will be noted at the beginning of each step.

If you are using a UNIX emulator like EUNICE, it is invoked from within the DCL Mode by typing the UNIX command, as shown in Fig. 10-1. The UNIX

```
$ UNIX

                              ***   EUNICE BSD Version 4.2    ***

% logout
$
```

Fig. 10-1. Executing the UNIX command to invoke the EUNICE operating system.

emulator prompt is the percent sign. To exit UNIX and return to DCL, type "logout" (in lowercase letters). If you are using a native mode UNIX system, then you will be in "UNIX Mode" when you log in on the VAX. In this case, the prompt is probably the dollar sign instead of the percent sign. Examples in these chapters use the percent sign, so just read the dollar sign wherever you see the percent sign in the figures.

LOGGING IN TO UNIX

When you "login" to UNIX, you might get more information on your screen than is shown in Fig. 10-1. If you get a message about copying some UNIX files, you might want to log out of UNIX first, and create a special subdirectory for this purpose (see Chapter 8).

If you are running a UNIX emulator like EUNICE, follow the example in Fig. 10-2 to see that your UNIX emulator really does run "on top" of the VMS operating system. (Skip this if you are running a native mode UNIX system.) The UNIX command **vms** is used to indicate DCL commands, to be executed by the VMS operating system.

```
$ UNIX

                                ***   EUNICE BSD Version 4.2   ***

% vms SHOW TIME
   4-AUG-1986 17:09:14
% vms SHOW USERS
            VAX/VMS Interactive Users
              4-AUG-1986 17:09:23
     Total number of interactive users = 6

  Username      Process Name       PID      Terminal
  BROWN         GURU             00001741   TTA6:
  CHANCE        FIFTY_FIFTY      0000174C   TTB7:
  GORBACHEV     BIG_RED          000017C5   RTA2:
  HUBBARD       A_HUBBARD        00001744   RTA1:
  REAGANRON     THE_GIPPER       000017CB   TTA1:
  THATCHERM     PM               000017C8   TXA6:
%
```

Fig. 10-2. Using the **vms** command to execute DCL commands from within UNIX.

The UNIX Prompt

You should now be in UNIX Mode. The percent sign is used to indicate the UNIX prompt. In native mode UNIX, this would be a dollar sign. By now, you should have realized that all UNIX commands must be typed in lowercase letters. The VMS option of using either uppercase or lowercase is not allowed in UNIX.

Fundamental UNIX Commands

As you learn to use different operating systems (e.g. VMS, UNIX, MVS, CP/M, PC-DOC, etc.), you will discover that certain fundamental commands are available in every system. Once you have learned the command language for one system, it is fairly easy to translate it to another system. A brief VMS-to-UNIX

dictionary is shown in Fig. 10-3. Try some of these UNIX commands now. Remember to type everything in lowercase letters. And, if you get stuck and can't get out of UNIX, try typing <CTRL/C>, <CTRL/D>, <CTRL/Y>, or <CTRL/Z>.

VMS	UNIX
COPY	cp
CREATE/DIRECTORY	mkdir
DELETE	rm, rmdir
DIRECTORY	ls
EDIT	ed
HELP	help
MAIL	mail
RENAME	mv
SEARCH	grep
SET DEFAULT	cd
SET PROTECTION	chmod
SET TERMINAL	stty
SHOW DEFAULT	pwd
SHOW DEVICE	df
SHOW TIME	date
SHOW USERS	who
SHOW TERMINAL	tty
SORT	sort
TYPE	cat, pr

Fig. 10-3. Some common DCL commands and their UNIX equivalents.

FILE PROCESSING WITH UNIX

In order to do some serious file processing, you will need some serious files to process. Figure 10-4 shows how to use the UNIX Text Editor to create a file. Follow this example now. If you make some typing mistakes, just leave them in for now; they can be corrected later.

Chapter 11 will investigate the details of the **ed** editor. For now, it is simply to be used to create some files. This use of the **ed** command is analogous to the ED command in the VAX/VMS DCL.

```
% ed
a
When to the sessions of sweet and silent thought
I summon up remembrance of things past.
I sigh the lack of many a thing I sought,
And with old woes new wail my dear times' waste.
.
w Shakespeare
180
q
%
```

Fig. 10-4. Using the text editor to create a file.

The first noticeable difference between the VMS EDT Editor and the UNIX **ed** Editor is that the latter has no prompt. After you type "ed," the cursor just sits at the beginning of the next line. This is typical UNIX understatement: it assumes that you know you are in the **ed** Mode, and that you know how to give some command. The command that you need to use is **a**. This stands for "append," which means you want to append text to the current contents of the editing buffer. Of course, the buffer is empty, so anything you add will actually become the contents of the new file.

The two UNIX commands **ed** and **a** together are equivalent to the VMS commands EDIT and INSERT. Now, whatever you type will be put into the editing buffer. When you finish typing your text, press <RET>, and then type a period, followed by <RET>, on a separate line. This is similar to <CTRL/Z> in EDT Mode: it terminates the **a** Mode, and returns you to the **ed** Mode.

The ed Editor

The UNIX Editor (like the BASIC Mode) does not require you to specify a filename when you enter the editor. Consequently (also like the BASIC Mode), it is possible to exit from the editor without saving a copy of your work.

The **ed** command to save the contents of the editing buffer is **w** (for "write"). By typing "w Shakespeare," the contents have been saved in a new file, named "Shakespeare."

The **ed** Editor responds to the **w** command by telling you how many characters were saved in the file specified. In this example, the editor saved 180 characters in the file named Shakespeare.

The command to exit from the Editor is **q**, for "quit." After executing that command, you should be back in the UNIX Mode.

Repeat the process, to create a Pascal file, as shown in Fig. 10-5. Notice that the UNIX command **ls**, which stands for "list," lists the files in alphabetical order, just as the VMS command DIRECTORY does. Finally, create one more file, named RUSSELL, as shown in Fig. 10-6.

```
% ls
Shakespeare
% ed
a
Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.
.
w Pascal
102
q
% ls
Pascal          Shakespeare
%
```

Fig. 10-5. Executing the **ls** command to list the files in a directory.

```
% ed
a
Thought is subversive and revolutionary,
destructive and terrible;
thought is merciless to privilege,
established institutons, and comfortable habit.
Thought looks into the pit of hell and is not afraid.
Thought is great and swift and free,
the light of the world, and the chief glory of man.
.
w Russell
293
q
% ls
Pascal          Russell          Shakespeare
%
```

Fig. 10-6. Creating a third file.

Options

In the VMS command languages (DCL), most commands can be modified by means of command qualifiers. For example, the /DATE qualifier attached to the DIRECTORY command causes the date and time at which the files were created to be displayed, along with the list of files. In DCL, all command qualifiers must be preceded by the slash character. Similarly, in the UNIX, Shell commands may be modified by means of *options*. These are symbols which are typed on the same line with the UNIX command, and preceded by a hyphen. For example, give the ls command with the -t option, as shown in Fig. 10-7. The -t option on the ls command requests that the files be listed in order of most recent change; "t" is for "time."

Figure 10-7 also illustrates the -l option on the ls command. This provides a "long" listing of files. It is similar to the /DATE/SIZE/OWNER/PROTECTION combination of qualifiers on the DIRECTORY command in DCL. The first sym-

```
% ls -t
Russell         Pascal          Shakespeare
% ls -l
total 3
-rwxr--r-- 1 reaganron   106 Aug 10 11:39 Pascal
-rwxr--r-- 1 reaganron   308 Aug 10 11:43 Russell
-rwxr--r-- 1 reaganron   182 Aug 10 11:33 Shakespeare
% ls -lt
total 3
-rwxr--r-- 1 reaganron   308 Aug 10 11:43 Russell
-rwxr--r-- 1 reaganron   106 Aug 10 11:39 Pascal
-rwxr--r-- 1 reaganron   182 Aug 10 11:33 Shakespeare
```

Fig. 10-7. Executing the ls command with the -t and -l options.

bols displayed for each file are the protection codes. (See Chapter 8, and Exercise 6 at the end of this chapter.) Here, "rwxr--r--" means that you, the owner, can read, write, and execute the file, while others can only read the file. The other information on each line includes the owner of the file, the size (in bytes) of the file, and the date and time it was last changed.

Another option available with the ls is the -r option, which reverses the listing order which would otherwise be used. Figure 10-8 illustrates the use of this option in various combinations.

```
% ls -r
Shakespeare      Russell          Pascal
% ls -rt
Shakespeare      Pascal           Russell
% ls -trl
total 3
-rwxr--r-- 1 reaganron   182 Aug 10 11:33 Shakespeare
-rwxr--r-- 1 reaganron   106 Aug 10 11:39 Pascal
-rwxr--r-- 1 reaganron   308 Aug 10 11:43 Russell
% ls Pascal
Pascal
% ls -l Pascal
-rwxr--r-- 1 reaganron   106 Aug 10 11:39 Pascal
% ls Sh*
Shakespeare
% ls *l
Pascal           Russell
% ls *l?
Russell
%
```

Fig. 10-8. Executing the **ls** command with the -t, -l, and -r options and with the * and ? wildcards.

Arguments

Figure 10-8 also illustrates the use of *arguments* with UNIX commands. In the following command:

```
% ls -1 Pascal
```

the argument is the filename Pascal. An argument in UNIX is the same as a parameter in VMS. The analogous DCL command would be:

```
$ DIRECTORY/SIZE/DATE/PROTECTION/OWNER Pascal.
```

UNIX Wildcards

Finally, notice how wildcard symbols are used in UNIX. The asterisk has the same meaning as in VMS; it stands for any substring of any length, including the empty string (length zero). The other wildcard symbol is the question mark. It works the same way as the percent sign in VMS; it stands for any single character.

```
% cat Pascal Russell
Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.
Thought is subversive and revolutionary,
destructive and terrible;
thought is merciless to privilege,
established institutons, and comfortable habit.
Thought looks into the pit of hell and is not afraid.
Thought is great and swift and free,
the light of the world, and the chief glory of man.
% cat S*
When to the sessions of sweet and silent thought
I summon up remembrance of things past.
I sigh the lack of many a thing I sought,
And with old woes new wail my dear times' waste.
%
```

Fig. 10-9. Executing the **cat** command.

CONCATENATING FILES

The UNIX command that is analogous to the DCL command TYPE is cat. This is an abbreviation for "concatenate," which means to string text together end-to-end. Follow the example in Fig. 10-9 to see how this works. Whatever files are listed after cat are displayed end-to-end, without any identification or separation. If you want the files to be identified and separated, then use the pr command (for "print") as in Fig. 10-10.

```
% pr *l

Aug 10 11:39 1986  Pascal Page 1

Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.




 Aug 10 14:21 1986  Russell Page 1

 Thought is subversive and revolutionary,
 destructive and terrible;
 thought is merciless to privilege,
 established institutons, and comfortable habit.
 Thought looks into the pit of hell and is not afraid.
 Thought is great and swift and free,
 the light of the world, and the chief glory of man.
```

Fig. 10-10. Executing the **pr** command.

The pr command was designed to be used with a printer. Therefore, it starts each new file on a new page, and it labels each page with the date, time, filename, and page number. Pages are normally 66 lines long (11 inches, 6 lines per inch), so you will get a lot of blank lines scrolling on your screen when you have short files.

The original purpose of the cat command was to concatenate files; that is, to string together several files into a single new file. Fig. 10-11 shows how to do this. The right angle bracket (>) is used to indicate output. In VMS, this would be achieved by the following DCL command:

```
$ TYPE/OUTPUT=THOUGHT.  S*., P*., R*.
```

Of course, it could also be done using the COPY command in DCL:

```
$ COPY S*., P*., R*.  THOUGHT
```

If you are running a native mode UNIX system instead of a UNIX emulator like EUNICE, go to the next section. Otherwise, return now to VMS and look at your directory. It should look like Fig. 10-12.

You are probably wondering, "Where did all those dollar signs come from?" Using dollar signs is a way for the VMS to remember that you want the first letter of each filename to be capitalized when you are running EUNICE. Because everything in UNIX is normally written in lowercase, the VAX uses its favorite symbol ($) to delineate substrings which are to be in uppercase.

```
% cat S* P* R* > Thought
% ls
Pascal              Russell          Shakespeare      Thought
% pr Th*

Aug 10 14:55 1986  Thought Page 1

When to the sessions of sweet and silent thought
I summon up remembrance of things past.
I sigh the lack of many a thing I sought,
And with old woes new wail my dear times' waste.
Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.
Thought is subversive and revolutionary,
destructive and terrible;
thought is merciless to privilege,
established institutons, and comfortable habit.
Thought looks into the pit of hell and is not afraid.
Thought is great and swift and free,
the light of the world, and the chief glory of man.
%
```

Fig. 10-11. Using the output symbol (>) to concatenate three files.

```
% logout
$ DIR

Directory SYS$USER:[REAGANRON.UNIX.TEST]

$P$ASCAL.;1          $R$USSELL.;1       $S$HAKESPEARE.;1     $T$HOUGHT.;1

Total of 4 files.
$ REN $T$HOUGHT. THOUGHT.
$ DIR

Directory SYS$USER:[REAGANRON.UNIX.TEST]

$P$ASCAL.;1          $R$USSELL.;1       $S$HAKESPEARE.;1     THOUGHT.;1

Total of 4 files.
$ UNIX
Last login: Sun Aug 10 14:46:14 on tty40

                        ***   EUNICE BSD Version 4.2    ***

% ls
Pascal          Russell         Shakespeare     thought
% mv thought Thought
% ls
Pascal          Russell         Shakespeare     Thought
%
```

Fig. 10-12. Executing the **mv** command. This example also shows how EUNICE names the files in VAX/VMS.

Return to UNIX and give the ls command to see that removing the dollar signs from around the T in THOUGHT causes UNIX to regard it as a lowercase t. Use the mv command to restore the capital T. This is the UNIX equivalent of the RENAME command; it stands for move, which suggests that the file has been "moved" to a new name.

Notice that the files created in UNIX were given no filetypes, so VMS assigned the empty string for each file type. The UNIX filename "Pascal" became the VMS filespec PASCAL.;1.

THE grep COMMAND

By now, you probably agree that most UNIX command names look a bit

```
% grep Thought Russell
Thought is subversive and revolutionary,
Thought looks into the pit of hell and is not afraid.
Thought is great and swift and free,
% grep thought Thought
When to the sessions of sweet and silent thought
thought is merciless to privilege,
%
```

Fig. 10-13. Executing the **grep** command.

```
% grep -n thought Thought
1:When to the sessions of sweet and silent thought
10:thought is merciless to privilege,
% grep -c s Thought
12
% grep -v s Thought
therefore, endeavor to think well,
the light of the world, and the chief glory of man.
%
```

Fig. 10-14. Executing the **grep** command with the -n, -c and -v options.

strange, especially after having learned DCL. One of the strangest UNIX command names executes one of its most useful commands. This is the grep command. (The name is an acronym for "global/regular-expression/print," which probably doesn't tell you much.) It is used to search for strings in files. Follow the example in Fig. 10-13 to see how it works. Notice that lowercase letters are distinguished from uppercase letters.

Figure 10-14 shows some of the options that are frequently used with the grep command.

The discussion in this chapter, has barely begun to examine the wealth of commands and options available in UNIX. You should try the rest of the commands listed in Fig. 10-3. Then, after working through the Self-Text Exercises, you should study some of the excellent UNIX books listed in the Bibliography.

SELF-TEST EXERCISES FOR CHAPTER 10

1. Investigate the sort command. Give the following commands:

   % sort Russell
   % sort -r Russell
   % sort -f Russell
   % sort +2 Russell
   % sort +3 Russell
   % sort -rf +2 Russell

   Then, try to guess what the options -r, -f, and + mean. (HINT: the "f" stands for "fold.")
2. Investigate the wc command:

   % wc *

   Try to guess what this command does. (HINT: the letters "w" and "c" abbreviate what is done to produce the second column of numbers.) There is no equivalent DCL command.
3. Investigate the tail command. Give the following commands:

   % tail Thought
   % tail -3 Thought

```
% tail +3 Thought
```

Try to guess what this command does.

4. The standard UNIX commands for using subdirectories are pwd, mkdir, rmdir, and cd. These are analogous to the DCL commands SHOW DEFAULT, CREATE/DIRECTORY, DELETE, and SET DEFAULT, respectively. Use these UNIX commands to check your current directory, create a subdirectory, move into it, check your current directory, move out of it, and delete it. (Refer to Chapter 8.)
5. Review Chapter 8 and the previous exercise, then give the following two commands:

```
% cd /
% ls
```

You have reached your UNIX root directory, which is the "father" of your own root directory. The ls command shows at least six subdirectories. Investigate the "bin" subdirectory with the following commands:

```
% cd bin
% ls
```

The files you see here with the .exe file types are the programs which carry out the UNIX commands! For example, when you give the who command, the program who.exe runs.

6. Investigate the chmod command. (See Appendix G.) This command allows you to change the protection code on a file. (See Chapter 8.) Use ls -l Pascal, for example, to read the protection code on the Pascal file. In UNIX, the categories are owner, group, and world. In each category there are three "switches": read access, write access, and execute access. The letters r, w, and x, mean that that access is allowed in the respective category; a hyphen means that that access is denied. The chmod uses a three-digit number whose digits in binary indicate which access is allowed. For example, the binary for the three digits 7 5 4 is 111 101 100, which would set the protection code rwx r-x r- -, allowing only read and access to the group, and only read access to the world. Determine what each of the following commands does:

```
chmod 741 Pascal
chmod 520 Pascal
chmod 700 *
```

7. Create a file named Descartes and insert the line "Cogito ergo sum." into it. This file will be used in Chapter 11.
8. Investigate the cal command. For example, type cal 11 1963. Then, use it to determine the day on which you were born.

Chapter 11

The UNIX Text Editor

Things should be made as simple as possible, but not simpler.

Albert Einstein

As in VAX/VMS, the most common way to create files in UNIX is to use a text editor. The standard text editor in VMS is called EDT. The standard text editor in UNIX is called ed. This editor was used in Chapter 10 to create some test files. In this chapter, the ed text editor will be discussed more thoroughly.

You saw in Chapter 10 that a file created in the ed editor is not automatically saved. Unless the w command (for write) is used, the contents of the editing buffer will be lost when you exit from the editor. In Chapter 10, you used the command w Shakespeare to save the Shakespeare file. The name had to be specified with the w command, because it wasn't specified anywhere else. The alternative is to specify the file when the editor is invoked with the ed command. If you do this, then you can save the file simply by pressing w without mentioning its name again. This is more like the way the EDT editor works.

In Fig. 11-1 a file containing the words of Mao Tse-Tung has been created. The ed editor responds to the "ed Mao_Tse-Tung" command with: ?Mao_Tse-Tung. This is analogous to the response "Input file does not exist," which the EDT editor gives when you invoke it to create a new file.

Notice that both the underline and the hyphen were used in the filename Mao_Tse-Tung. This is allowed in both UNIX and in VAX/VMS. Recall, from

from Chapter 2, that VMS allows letters digits, the dollar sign, the underline, and the hyphen in its filenames, which may contain up to 39 of these characters. UNIX allows only 14 characters for complete file specifications (it will truncate longer filespecs). It does, however, allow the use of almost every possible character in a filename. The only characters not allowed are those which would cause serious problems if used, for example, the blank (space) character, and the angle bracket characters. Of course, a filename like "-lr" would be unwise.

```
% ed Mao_Tse-Tung
?Mao_Tse-Tung
a
The policy of letting a hundred flowers blossom
and a hundred schools of thought contend
is designed to promote the flourishing of the arts
and the progress of science;
it is designed to enable a socialist culture
to thrive in our land.
.
w
237
q
%
```

Fig. 11-1. Executing the **a**, **.**, **w**, and **q** commands in the UNIX text editor.

THE LINE EDITOR

The UNIX text editor is a line editor; it regards a block of text as a sequence of numbered lines, and most commands are verbs which refer to the numbered lines. This is very similar to the EDT Line Editor. However, in the ed editor, the lines are always numbered 1, 2, 3, etc.; they are resequenced automatically after any command which rearranges them. To see how the line numbers work, invoke the editor again with your Mao_Tse-Tung file and follow the example in Fig. 11-2. Notice that when you invoke ed with an existing file, it tells you how many characters the file has.

```
% ed Mao_Tse-Tung
237
1,$p
The policy of letting a hundred flowers blossom
and a hundred schools of thought contend
is designed to promote the flourishing of the arts
and the progress of science;
it is designed to enable a socialist culture
to thrive in our land.
```

Fig. 11-2. Using the **p** command to display the file.

Commands

Commands refer to line numbers in the UNIX editor as they do in the VMS EDT Line Editor (see Chapter 2): the dot (or period) always denotes the current line, and the minus and plus signs are used to indicate lines before and after other lines. Instead of BEGIN and END, however, UNIX uses the number one and a dollar sign to indicate the first and last lines in the buffer. (The first line will always be numbered 1, because the lines are always resequenced automatically.) UNIX uses the comma instead of the colon to represent a block of lines, called a *range*. For example, "3,6" refers to lines three through six. A single number means the range containing only that one line. Thus, "6" is the same as "6,6". If you type a command without any line numbers, ed assumes that you are referring to the current line.

Syntax

Consistent with UNIX's obsession with brevity, all ed commands are single letters. (See Appendix H for a list of them.) If a command refers to a range of lines, the range is specified first, and then the one-letter command follows. For example, 1,$p means to print (display) all the lines of the file. The line numbers are not displayed.

CREATING A FILE

You saw in Chapter 10 that the command w means to save (write) the file. This write command is actually the same as the WRITE command in the VMS

```
f
Mao_Tse-Tung
w Mao#1-6
237
f
Mao_Tse-Tung
1,$p
The policy of letting a hundred flowers blossom
and a hundred schools of thought contend
is designed to promote the flourishing of the arts
and the progress of science;
it is designed to enable a socialist culture
to thrive in our land.
1,4w Mao#1-4
169
q
% ls
Mao#1,4         Mao_Tse-Tung    Pascal          Shakespeare     Thought
Mao#1-7         Descartes       Russell
% cat Mao#1-4
The policy of letting a hundred flowers blossom
and a hundred schools of thought contend
is designed to promote the flourishing of the arts
and the progress of science;
%
```

Fig. 11-3. Using the w command to create subfiles.

EDT editor (see Appendix C). Used alone, it saves the entire buffer in a file whose filename was specified when the editor was invoked. Alternatively, you may command that the buffer be saved in a new file, simply by using w with the new filename. Finally, if you precede the command with a line range, then only those lines will be saved.

Follow the example in Fig. 11-3. First, the f command is given to ascertain the current filename. Then, the current contents of the buffer is written into a new file named Mao#1-6. The f command is used again to see that the current filename is left unchanged, and the 1,$p command (which is equivalent to EDT's TYPE WHOLE command) is given to see that the buffer is also left unchanged.

The next command, 1,4w Mao#1-4, saves only lines one through four into a new file named Mao#1-4. After leaving ed, the directory is listed.

Notice that the directory command lists the files in columns. In this example, the order (most recently accessed) is: Mao#1-4, Mao#1-6, Mao_Tse-Tung, Descartes, Pascal, Russell, Shakespeare, and Thought. Finally, the cat command is used to see that the new Mao#1-4 file really does contain the first four lines of

```
% ed Pascal
102
1,$p
Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.
1r Descartes
17
1,$p
Thought makes the whole dignity of man;
Cogito ergo sum.
therefore, endeavor to think well,
that is the only morality.
.
that is the only morality.
r Descartes
17
1,$p
Thought makes the whole dignity of man;
Cogito ergo sum.
therefore, endeavor to think well,
that is the only morality.
Cogito ergo sum.
Or Descartes
17
1,$p
Cogito ergo sum.
Thought makes the whole dignity of man;
Cogito ergo sum.
therefore, endeavor to think well,
that is the only morality.
Cogito ergo sum.
```

Fig. 11-4. Using the r command to read (insert) an external file into the editing buffer.

the original Mao_Tse-Tung file. Incidentally, if you type something which ed cannot understand, it will respond with a question mark.

Inserting External Files

Return to ed as shown in Fig. 11-4. In this example, you will use the Pascal file created in Chapter 10. (If you don't have this file, insert it now using the append command, as shown in Fig. 10-5. Also create the Descartes file as described in Self-Test Exercise 7 at the end of Chapter 10.)

```
1,$p
Cogito ergo sum.
Thought makes the whole dignity of man;
Cogito ergo sum.
therefore, endeavor to think well,
that is the only morality.
Cogito ergo sum.
3,5d
1,$p
Cogito ergo sum.
Thought makes the whole dignity of man;
Cogito ergo sum.
2i
Computo ergo sum
.
1,$p
Cogito ergo sum.
Computo ergo sum.
Thought makes the whole dignity of man;
Cogito ergo sum.
```

Fig. 11-5. Using the **d** command to delete lines.

The r command (for "read") is the opposite of the w command. It is used to read (insert) an external file into the editing buffer. It has the same effect as the INCLUDE command in EDT. (See Self-Test Exercise 10 at the end of Chapter 3.) In this example, the Descartes file has been read into the buffer at line one. Notice that the insertion is made below the line specified. (EDT inserts the file above the line specified.)

If you omit the line number before the r, the file will be inserted below the current line. Type the dot to see what the current line is. In the example above, the current line is the last line in the buffer, because that was the last line referenced in the last command (which was the TYPE WHOLE command 1,$p).

If you want to insert a file into the beginning of the editing buffer, above line one, then use 0r to read it to the position below line zero. Line zero is the imaginary line above the first line of text in the buffer (just as END is the imaginary line below the last line of text in the EDT buffer).

DELETING AND INSERTING LINES

Deleting and inserting lines in ed is similar to writing and reading files. The commands to be used are d and i. Follow the example in Fig. 11-5.

The command 3,5d deletes lines three through five. The command 2i allows the insertion of line(s) above line two. Note that the insertion command inserts text above the reference line, while the read command inserts text below the reference line.

The insert command is actually very similar to the append command. In each case, you move into a inner "mode," where everything you type is inserted in the editing buffer (just like the INSERT Mode in EDT). You leave the mode by typing a period alone on a separate line.

```
1,$p
Cogito ergo sum.
Computo ergo sum.
Thought makes the whole dignity of man;
Cogito ergo sum.
1,4s/gi/mpu/
1,4p
Computo ergo sum.
Computo ergo sum.
Thought makes the whole dignity of man;
Computo ergo sum.
2,4s/mpu/gi/
1,4p
Computo ergo sum.
Cogito ergo sum.
Thought makes the whole dignity of man;
Cogito ergo sum.
1,3s/it//
1,4p
Computo ergo sum.
Cogo ergo sum.
Thought makes the whole digny of man;
Computo ergo sum.
```

Fig. 11-6. Using the s command to substitute (replace) parts of the buffer.

The Substitute Command

The substitute command in ed is essentially the same as the SUBSTITUTE command in EDT. The only difference is that, as with all ed commands, the line number range is specified first. Notice in Fig. 11-6 that, unlike EDT, ed does not tell you how many substitutions have been made. Extra care is needed, therefore, with this command.

To see how the move and transfer commands work, first empty your buffer and create the six test lines as indicated in Fig. 11-7. Notice that the response to your trying to print all the lines of an empty file is a question mark, because at that point, there is no line 1.

```
1,4d
1,$p
?
a
Line 1.
Line 2.
Line 3.
Line 4.
Line 5.
Line 6.
.
1,$p
Line 1.
Line 2.
Line 3.
Line 4.
Line 5.
Line 6.
1,3m5
1,6p
Line 4.
Line 5.
Line 1.
Line 2.
Line 3.
Line 6.
2,3t0
1,$p
Line 5.
Line 1.
Line 4.
Line 5.
Line 1.
Line 2.
Line 3.
Line 6.
```

Fig. 11-7. Using the m and t commands to move and transfer (copy) parts of the buffer.

The Move Command

The m command does the same in ed that the MOVE command does in EDT. The only differences are that the line range to be moved is specified first, and the position to which they are moved is below the target line specified. In this example, lines one through three are moved to the position below line five.

The t command is similar to the m command, but it does not delete the range of lines from their original position. It has the same effect as the COPY command in EDT.

The Search Command

To test the search command, clear the buffer again, and then retrieve the Thought file created in Chapter 10. Type "1" to make the first line the current line. Then, type "/Thought/" as shown in Fig. 11-8. You can repeat the command by typing it again, or, more simply, by pressing the up-arrow cursor control

```
1,$d
r Thought
572
1,$p
When to the sessions of sweet and silent thought
I summon up remembrance of things past.
I sigh the lack of many a thing I sought,
And with old woes new wail my dear times' waste.
Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.
Thought is subversive and revolutionary,
destructive and terrible;
thought is merciless to privilege,
established institutons, and comfortable habit.
Thought looks into the pit of hell and is not afraid.
Thought is great and swift and free,
the light of the world, and the chief glory of man.
1
When to the sessions of sweet and silent thought
/Thought/
Thought makes the whole dignity of man;
/Thought/
Thought is subversive and revolutionary,
/Thought/
Thought looks into the pit of hell and is not afraid.
```

Fig. 11-8. Searching the buffer for a string.

```
1
When to the sessions of sweet and silent thought
/sought/,/thought/p
I sigh the lack of many a thing I sought,
And with old woes new wail my dear times' waste.
Thought makes the whole dignity of man;
therefore, endeavor to think well,
that is the only morality.
Thought is subversive and revolutionary,
destructive and terrible;
thought is merciless to privilege,
thought is merciless to privilege,
1,$p
When to the sessions of sweet and silent thought
I summon up remembrance of things past.
I sigh the lack of many a thing I sought,
Thought is subversive and revolutionary,
destructive and terrible;
thought is merciless to privilege,
established institutions, and comfortable habit.
Thought looks into the pit of hell and is not afraid.
Thought is great and swift and free,
the light of the world, and the chief glory of man.
```

Fig. 11-9. Using the search specifier to restrict the scope of the **p** and **d** commands.

key. (As in all VAX modes, this displays the last command executed.) Note that, unlike the EDT search command, uppercase and lowercase are distinct in **ed**. The search skips over line nine, for example, which contains "thought."

As in EDT, the search specifier can also be used to define line ranges. Follow the example in Fig. 11-9. First, all the lines from the first line containing "sought" to the next occurrence of a line containing "thought" are printed. Next, the dot command shows what the current line is. Finally, all lines from the sixth line above the current line to the next line after the one that contains "moral" are deleted. This deletes four lines.

SELF-TEST EXERCISES

1. Explain the difference between the add command and the insert command. Why can't the insert command be used to insert text into an empty buffer?
2. Give the equivalent delete and insert commands for the command: .-2,/it's/c.
3. Give the equivalent delete and transfer commands for the command: +,$m/Blaise/.

Appendices

Appendix A

VAX Modes

Though analogy is often misleading, it is the least misleading thing we have.

Samuel Butler

This appendix summarizes the relationships between the various modes described in the book. Figure A-1 shows these relationships graphically.

The main mode on the VAX is called the DCL Mode. Within this mode, five primary submodes have been described in this book: the EDT Line Mode, the Minitab Mode, the BASIC Mode, the UNIX Mode, and the MAIL Mode. The DCL commands which invoke these modes are EDIT, MINITAB, BASIC, UNIX, and MAIL, respectively. Some of these primary submodes themselves have submodes. For example, the Keypad Mode is a submode of the EDT Line Mode, and the SEND Mode is a submode of the MAIL Mode.

Every mode and submode is represented by a rectangle in Fig. A-1. The name of the mode is printed on top of the rectangle. The command that breaks the left side of the rectangle is used to invoke the mode represented by the rectangle. For example, CHANGE is the EDT command that invokes the Keypad Mode. The command which breaks the right side of the rectangle is the command that terminates the mode. For example, the command q terminates the edit Mode within the UNIX Mode, returning the process to the UNIX Mode. The prompt for each mode is shown in the upper left corner of the rec-

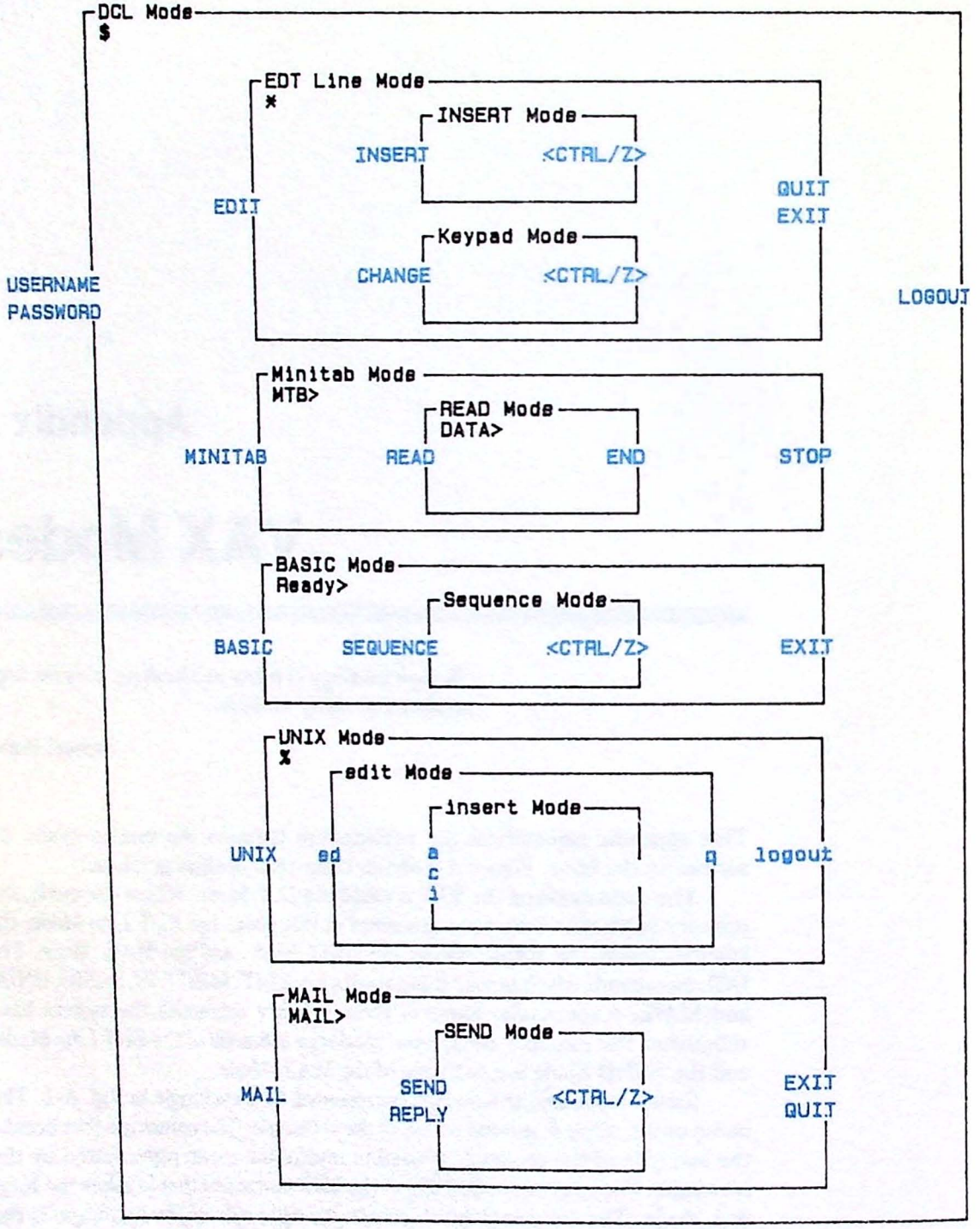

Fig. A-1. Vax Modes.

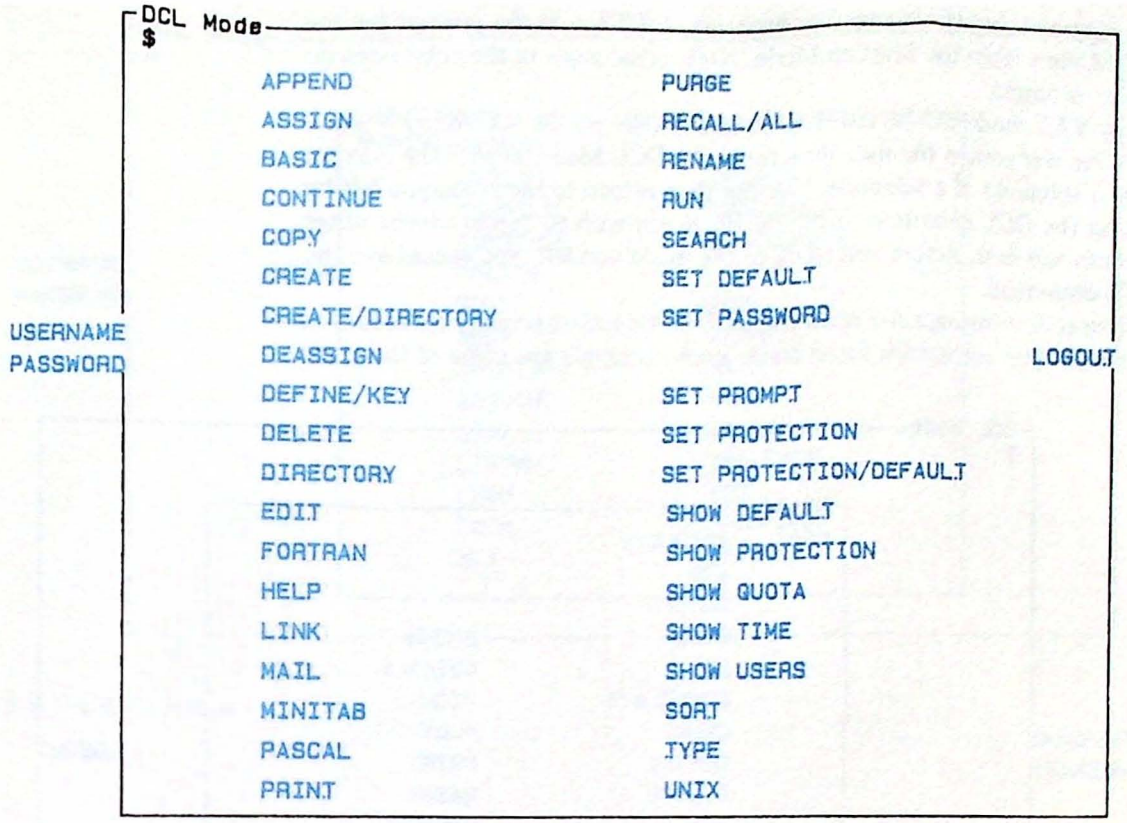

Fig. A-2. The DCL Mode.

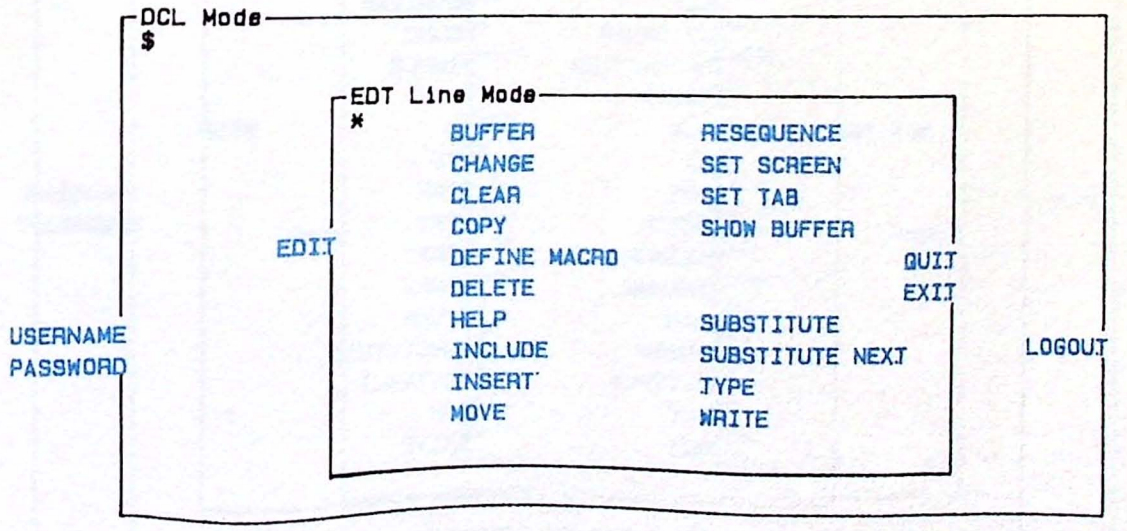

Fig. A-3. The EDT Line Mode.

tangle representing the mode. For example, DATA> is the prompt for the READ Mode within the Minitab Mode. Notice that some of the submodes do not have prompts.

Any VAX mode can be interrupted at any time by the <CTRL/Y> command. This will return the user directly to the DCL Mode, even if the current mode is a submode of a submode. You can then return to the mode you left by executing the DCL command CONTINUE. If you wish to execute some other DCL command first, before returning to the mode you left, you should use the SPAWN command.

Figures A-2 through A-7 show the DCL Mode and its primary submodes in more detail. The commands listed inside each rectangle are some of the com-

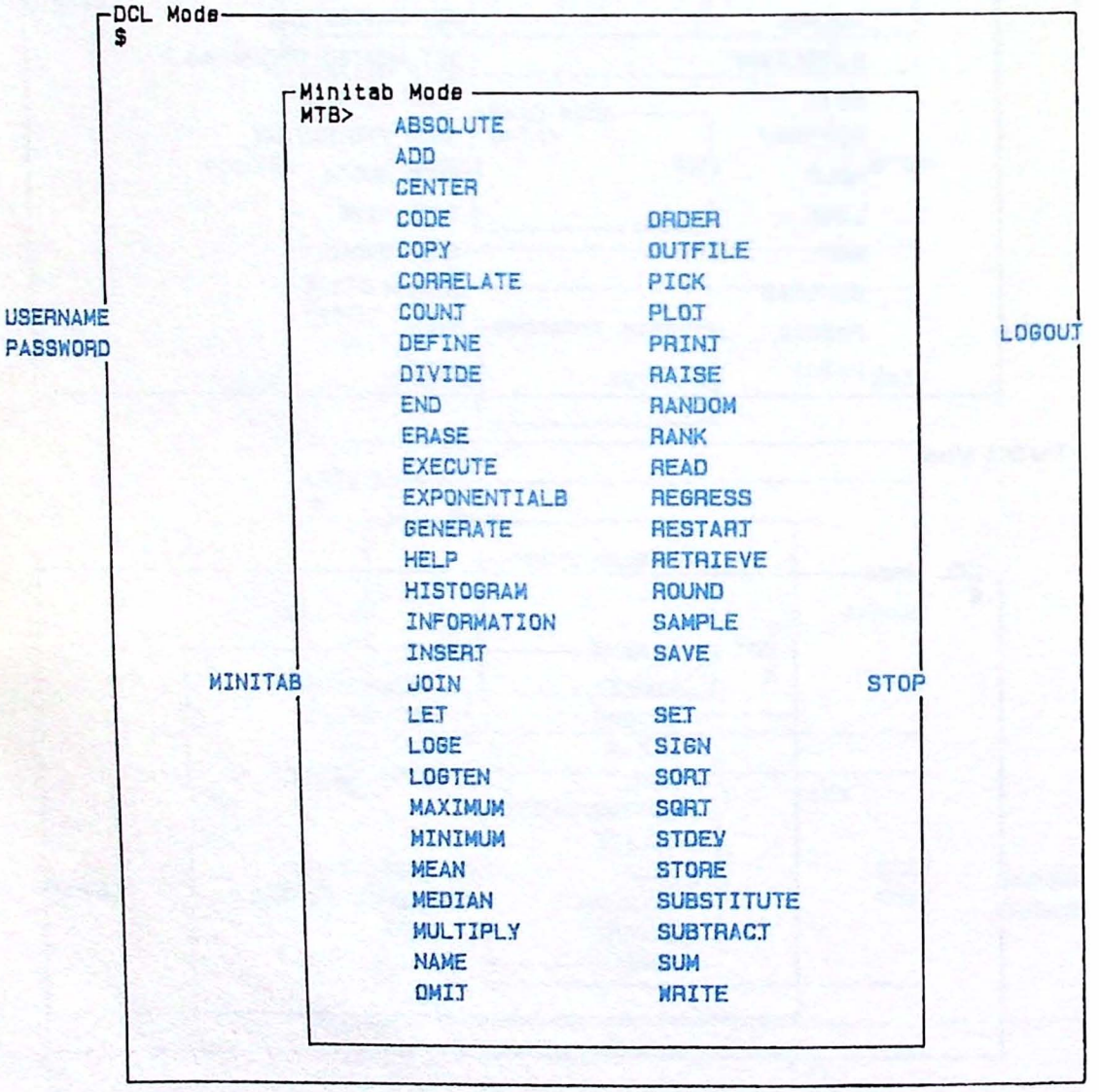

Fig. A-4. The Minitab Mode.

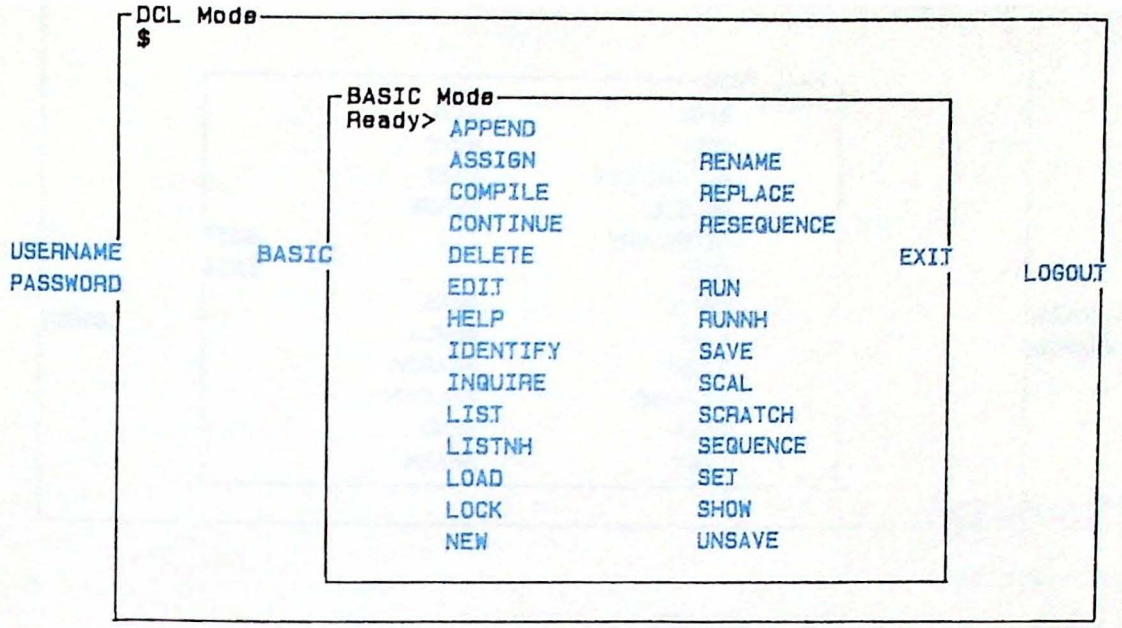

Fig. A-5. The BASIC Mode.

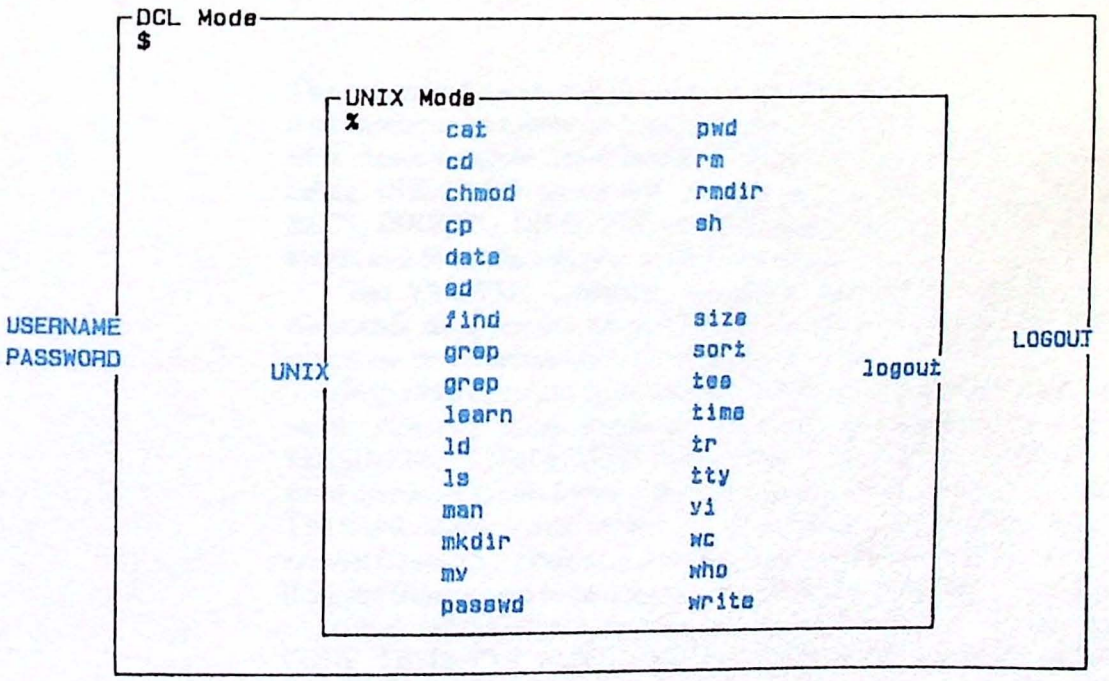

Fig. A-6. The UNIX Mode.

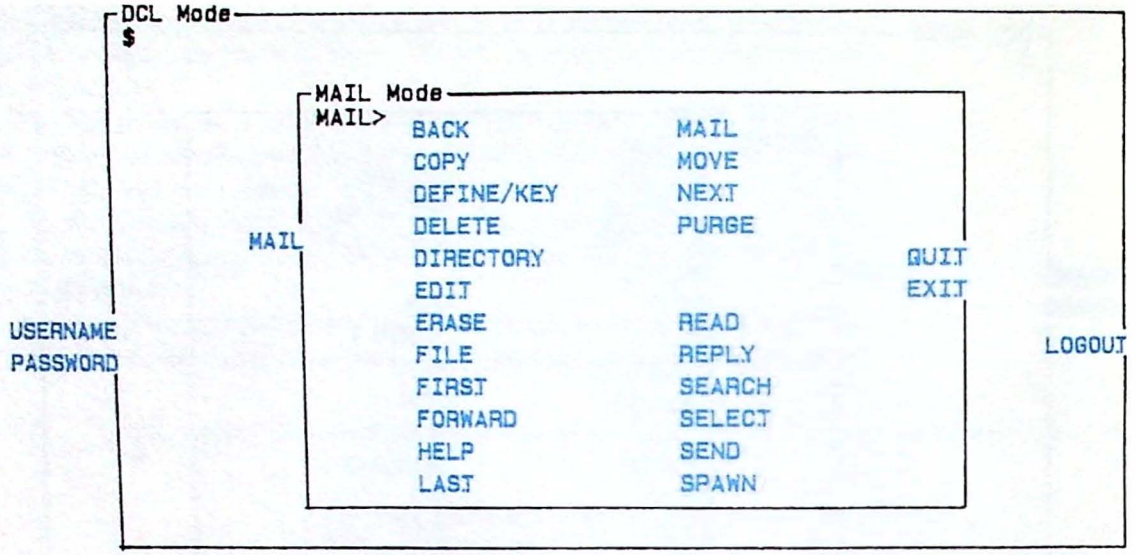

Fig. A-7. The MAIL Mode.

mands available in the mode represented. Notice that some commands, like COPY, DELETE, SEARCH, and SORT, are available in several different modes.

One command is available in every mode: the HELP command. The use of this command is one of the best ways to become familiar with a given mode. In particular, it will list the commands available in the mode.

Appendix B

DCL Commands

Don't ask what it means, but rather how it is used.

Wittgenstein

This appendix lists some of the most commonly used DCL commands. All DCL commands can be abbreviated to at most 4 letters. The minimal abbreviation for each command given below is indicated by colored letters. For example, the listing DIRECTORY means that the user may type DIR, DIRE, DIREC, DIRECT, DIRECTO, DIRECTOR, or DIRECTORY (or, of course, any of these in lowercase) to obtain a display of the directory.

The VAX/VMS Command Interpreter automatically converts all DCL commands to uppercase before interpreting them. Therefore, you may use uppercase or lowercase when you type the commands.

In the following descriptions, the word ***name*** stands for any filename, logical name, directory name, keyboard key name, or username; for example: PROGRAM1, SYS$OUTPUT, COMPROCS, PF1, or REAGANRON. The word *command* stands for any DCL command, for example, SHOW USERS. The word *filespec* stands for any file specification (e.g., TEST1.TXT). The symbol *filespec[, . . .]* indicates that more than one filespec may be listed, and, if so, the filespecs are to be separated by comas. For example, COPY *filespec[, . . .] filespec* indicates that both of the following are acceptable DCL commands: COPY TEMP.TXT NOTE.TXT, and COPY PART1.PAS,PART2.PAS, PART3.PAS PROGRAM.PAS.

Any DCL command may be interrupted by pressing <CTRL/C> or <CTRL/Y>. If <CTRL/Y> is used, then the execution of the command may be resumed by the CONTINUE command.

APPEND *filespec[,...] filespec*

Attaches the files listed first to the end of the last file specified.

Example: AP PART1.PAS,PART2.PAS PROGRAM.PAS

ASSIGN *filespec name*

Assigns the file specified first to the logical name specified second. This is useful in diverting input or output during the running of a program.

Example: ASSIGN NAMES.DAT BAS$INPUT

BASIC

Invokes the BASIC editor mode. In this mode, you can write and run BASIC programs just as you would on a microcomputer. It also includes immediate mode BASIC, wherein you can execute one-line BASIC programs without line numbers.

Example: BAS

BASIC *name*

Compiles the BASIC program found in the file with the given filename and with filetype BAS. This creates the object module, with the filetype OBJ.

Example: BAS PROGRAM1

CONTINUE

Resumes the execution of a DCL command, a VAX mode, or a program which was interrupted by pressing <CTRL/Y>.

Example: C

COPY *filespec[,...] filespec*

Creates the file specified last and copies the files specified first into it.

Example: COP HEADER.TXT,LINE*.TXT PARAGRAPH.TXT

CREATE *filespec[,...]*

Creates the files specified. The contents, followed by <CTRL/Z>, is typed for each file created.

Example: CR MEMO.TXT

CREATE/DIRECTORY [*.name*]

Creates a new subdirectory with name specified.

Example: CR/D [.MISC]

DEASSIGN *name*

Cancels the corresponding assignment made by the last ASSIGN command to the logical name specified.

Example: DEASSIGN BAS$INPUT

DEFINE/KEY *name* "command"

Redefines a keyboard key so that the command specified executes whenever the key is pressed.

Example: DEF/K PF1 "SHOW USERS"

DELETE *filespec[,...]*

Deletes all files specified. The version numbers must be specified.

Example: DEL FILE2.TXT;5, *.OBJ;*, TEST%.*;

DIRECTORY *filespec[,...]*

Lists all of the user's files in the current directory whose file specifications conform to those given. If no filespec is given, then all files in the current directory are listed. The following command qualifiers are often used with the DIRECTORY command: /DATE, /EXCLUDE=*(filespec[,...])*, /PROTECTION, /SIZE

Example: DIR/SIZ/EXC=(P1.*,P4.*) *.PAS,NOTE%.TXT

EDIT *filespec*

Invokes the EDT editor. If the editing session is terminated by the EXIT command, the contents of its MAIN buffer are saved in the file specified. If the command qualifier /RECOVER is included and the corresponding journal file is in the user's directory, the file will be restored.

Example: ED NOTE.TXT

FORTRAN *name*

Compiles the FORTRAN program found in the file with the given filename and with filetype FOR. This creates the object module, with the filetype OBJ.

Example: F PROGRAM3

HELP *command*

Displays information about the DCL command that is specified. The command qualifier /NOPAGE will cause continuous scrolling.

Example: HE HE

LINK *name*

Invokes the VAX/VMS Linker to resolve all external references in the object module (*name*.OBJ), creating the executable image (*name*.EXE).

Example: LIN PROGRAM3

LOGOUT

Terminates the current process. If no subprocesses exist, this command will disconnect you from the VAX.

Example: LO

MAIL *filespec name*

Invokes the VAX/VMS Personal MAIL Utility. If a file is specified, then that file is sent to the user specified. Otherwise, the MAIL Mode is invoked. (See Appendix I.) The qualifier /EDIT will allow you to use the EDT editor on the SEND and REPLY commands.

Example: MAIL PROGRAM1.TXT BUSHGEORG

MINITAB

Invokes the Minitab Utility (see Chapter 6 and Appendix E).

Example: MI

PASCAL *filename*

Compiles the Pascal program found in the file with the given filename and with filetype PAS. This creates the object module (*name*.OBJ).

Example: PAS PROGRAM2

PRINT *filespec[,...]*

Prints the files listed on the default system printer.

Example: PR PROGRAM4.FOR

PURGE *filespec[,...]*

Deletes all but the highest-numbered version of the file(s) specified. If no file is specified, then the entire directory is purged.

Example: PU *.OBJ,P1%.TXT

RECALL /ALL

Displays the last 20 DCL commands executed.

Example: REC/A

RENAME *filespec[,...] filespec*

Renames all the files listed first with the last name specified. If more than one file is given the same name and type, then successive version numbers are used.

Example: REN PROGRAM*.PAS PROGRAM.PAS

RUN *name*

Runs the executable image (*name*.EXE).

Example: RUN PROGRAM5

SET DEFAULT [*.name*]

Changes the default directory.

Example: SET DEF [.PROGRAMS.FORTRAN.OLD.1962]

SET PASSWORD

Allows you to change your password. The command qualifier /GENERATE will produce pronounceable nonsense words from which you can choose a new password.

Example: SET PA

SET PROMPT="*string*"

Changes the DCL prompt. If the string option is given then that becomes the prompt. Otherwise, the dollar sign ($) is restored.

Example: SET PROM="Yes?"

SET PROTECTION = (*protocode*) *filespec*[,. . .]

Changes the protection code on the files listed. (See Chapter 8.)

Example: SET PROT=(W:E,O:WRED) PROGRAM1.BAS

SET PROTECTION = (*protcode*)/DEFAULT

Change the default protection code for your current terminal session. (See Chapter 8.)

Example: SET PROT=(O:WRE,W:E)/DEF

SHOW DEFAULT

Shows your current default directory.

Example: SHO DEF

SHOW PROTECTION

Shows your current default protection.

Example: SHO PROT

SHOW QUOTA

Shows the amount of memory space you have been allocated for your current disk. Note that memory space is measured in terms of blocks. One block equals 512 bytes.

Example: SHO QUO

SHOW TIME

Shows the current date and time.

Example: SHO TI

SHOW USERS

Shows the usernames (and other data) of all the users who are currently logged in to the VAX.

Example: SHO US

SORT *filespec[,...] filespec*

Sorts the file specified in the first list and copies the resulting sorted records in the last file specified.

Example: SORT NEW.DAT TEMP.DAT

SPAWN *command*

Creates a subprocess. If another DCL command is included, then the subprocess terminates when the command has finished executing. Otherwise, the subprocess continues until it is terminated by the LOGOUT command.

Example: SP SHO TI

TYPE *filespec[,...]*

Displays the contents of the files specified.

Example: T *.DAT,PROGRAM1.BAS

UNIX

Invokes the UNIX emulator Eunice. (See Chapters 11 and 12.)

Example: UNIX

Appendix C

EDT Line Commands

Words differently arranged have a different meaning,
and meanings differently arranged have different effects.

Blaise Pascal

The EDT editor allows you to create and modify files. To invoke the editor, enter the EDIT command in the DCL Mode. For example, the command EDIT NOTE.TXT would allow you to edit the file NOTE.TXT. The EXIT command will save your file, exit from the editor and return you to the DCL Mode. (See Appendix A.)

This appendix lists the editing commands for the EDT Line Mode. All commands are written in uppercase (although the editor ignores the letters' case) and are preceded by the EDT prompt (an asterisk). The minimal abbreviation, if any, for each command is indicated by colored print. For example, the listing EXIT indicates that the editor will accept any of the following for the EXIT command: EXIT, exIT, EXI, ex.

The word *range* represents a reference to a block of lines in a buffer. If the reference begins with an equals sign (=), then the word which follows is the name of the buffer; otherwise, the block is understood to be in the current buffer. The remainder of the reference consists of either a single line or a range of lines. A range of lines is specified by the form x:y, where x and y are lines. A range of lines may also be specified with the word BEFORE (meaning BEGIN:.) or with the

word AFTER (meaning . : END). If a single line location is being specified by a range of lines, then the first line in that range is to be the line position. The following are examples of range references: 31:48, -'Ada':29, = TEMP . : E, and = PASTE 5.

The word *name* refers to a buffer or a macro. The word *command* refers to any EDT Line Mode command. The word *string* refers to any one-line string (without the enclosing apostrophes). The letter *n* refers to any positive integer.

The square brackets ([]) indicate an optional part of the command. Anything enclosed in square brackets may be omitted from the command, or may appear once. For example, DELETE [range] indicates that the DELETE command may be entered by itself or followed by a range.

CHANGE

Invokes the full-screen EDT Keypad Mode (also called the "Change mode"). This activates the cursor direction keys and the keypad editing keys. (See Chapter 4 and Appendix J.) EXCEPTION: if your terminal does not support keypad editing, then the CHANGE command puts you in Nokeypad Mode.

Example: *CHANGE

CLEAR *name*

Deletes the contents of the buffer named. The buffer need not be the current buffer. The emptied buffer remains defined for the rest of your editing session.

Example: *CLEAR PASTE

COPY [*range*] TO [*range*] [/DUPLICATE:*n*]

Copies the lines specified by the first range to the position specified by the second range. If either range is omitted, the default is the current line. If the /DUPLICATE:n qualifier is used, then the COPY is repeated n times.

Examples: *COPY 5:12 to=TEMP
*COPY=STARS TO /D:3

DEFINE MACRO *name*

Creates a buffer with the given name and designates that buffer as a macro. This means that only EDT commands should be inserted into the buffer. Then, whenever the name of the buffer is typed as an EDT command, the commands in the buffer will be executed in sequence. Use the INSERT commands to insert commands into the macro buffer.

Example: *DEFINE MACRO FIX

DELETE [*range*]

Deletes the range specified.

Examples: *DELETE .+2:'Wilson'
*DELETE PASTE 8:END
*DELETE = TEMP

EXIT [*name*] [/SAVE]

Terminates the editing session and returns you to the DCL Mode. The contents of the MAIN buffer are saved in a new version of the file you named when you began the session. If name is included, then that filename is used instead. If the /SAVE qualifier is included, then the journal file is not deleted.

Examples: *EXIT
*EXIT TEMP.TXT /SAVE

FIND *range*

Locates the line specified by range. It makes the first line of that range the current line. If the range specifies a buffer, then that becomes the current buffer. Notice that the word "FIND" may be omitted.

Examples: *FIND 23
*'Pascal,'
*=FIX 8
*=MAIN

HELP [*command*]

Displays information about the specified command. If the command is omitted, general information is displayed.

Examples: *HELP COPY
*HELP HELP

INCLUDE *name*[*range*]

Copies the contents of the file specified by name into the location specified by range, or into the current location if no range is specified.

Examples: *INCLUDE OLD_NAMES.DAT =TEMP
*INCLUDE HEADER.TXT BEGIN
*INCLUDE TYPEDEFS.PAS

INSERT [*range*] [;*string*]

Inserts the string typed into the range specified. If no range is specified, the string is inserted at the current position. If the semicolon is omitted, then the command invokes the Insert Mode within the EDT Line Mode. (See Appendix J.) Everything typed within the Insert Mode is inserted at the range specified. The insert Mode is terminated with <CTRL/Z>.

Examples: *INSERT =BUFFER2
*INSERT -'Babbage'

MOVE [*range*] TO [*range*]

Moves the lines in the first range specified to the position determined by the second range.

Examples: *MOVE REST TO 32
*MOVE =BUFFER6 10:END TO =TEMP

QUIT [/SAVE]

Terminates the editing session and returns you to the DCL Mode. The contents of the main buffer are lost. If the /SAVE qualifier is included, then the journal file is not deleted.

Example: *QUIT

RESEQUENCE [*range*]

Renumbers the lines so that they are numbered with consecutive integers. This command is useful after having used the COPY, MOVE, or INSERT commands.

Examples: *RESEQUENCE
*RESEQUENCE =B2 10:END

SET [NO]NUMBERS

SET NONUMBERS suppresses the display of line numbers. It is cancelled with the SET NUMBERS command.

Examples: *SET NONUMBERS
*SET NUMBERS

SET SCREEN *n*

Controls the maximum number n of characters displayed per line.

Example: *SET SCREEN 64

SET TAB *n*

Sets the number of tab spaces used in the Character Editing Mode. (See Chapter 4.)

Example: *SET TAB 2

SET [NO]TRUNCATE

The SET TRUNCATE command causes the lines to be truncated; the SET NOTRUNCATE causes them to be continued on the next line. This command is usually used with the SET SCREEN command.

Example: *SET TRUNCATE

SHOW BUFFER

Displays a list of all buffers in use together with their numbers of lines. It also indicates which is the current buffer.

Example: *SHOW BUFFER

SHOW SCREEN

Displays the current screen width set by the SET SCREEN command.

Example: *SHOW SCREEN

SUBSTITUTE/*string*/*string* [/ *range*] [/QUERY] [/NOTYPE]

Replaces the first string with the second at every occurrence within the range specified. If no range is specified, then only the first occurrence of the string in the current line is replaced. The /QUERY qualifier causes the editor to allow you to control each proposed substitution by asking you for a "Y" or "N" response. The /NOTYPE qualifier controls whether the affected lines are displayed.

Examples: *SUBSTITUTE/Cabbage/Baggage/BEGIN:.
*SUBSTITUTE/. /. /=TEMP/NOTYPE

[SUBSTITUTE] NEXT [/*string*/*string*/]

Replaces the first string with the second at its next occurrence found in the forward direction. If the strings are omitted, the strings given in the last execution of a SUBSTITUTE command are used.

Examples: *SUBSTITUTE NEXT/Blaise Pascal/Pascal/
*NEXT

[TYPE] [*range*] [/STAY]

Displays the range of lines specified. The /STAY qualifier causes the current line to be left unchanged.

Examples: *TYPE WHOLE
*5:12/STAY
*TYPE =PASTE

WRITE *name* [*range*]

Copies the contents of the specified range into the file specified by name. This is the inverse of the INCLUDE command, except that the entire buffer is copied if no line range is specified.

Examples: *WRITE OLD_NAMES.DAT =TEMP
*WRITE HEADER.TXT

' {apostrophe}

The "ditto" command, entered by the apostrophe ('), causes the last command executed to be executed again.

Example: '

Appendix D

EDT Keypad Commands

Here we have an opportunity of expounding more clearly what has already been said.

Descartes

When you invoke the EDT editor by means of the EDIT command, you are in the EDT Line Mode. To utilize full-screen character editing, you need to move from the Line Mode into the Keypad Mode. This is done with the CHANGE command.

The EDT character editing mode is called the Keypad Mode because the cluster of 18 keys which are used for the character editing is called the keypad. This cluster is shown in Fig. D-1.

In addition to the 18 keypad keys, the four cursor control keys, the <BS> key, the <LF> key, and the <CTRL> key are also used. The cursor control keys are labeled with arrows. On the VT100-type keyboards, they are on the top row above the <RET> key. On the VT200 series keyboards, they are at the bottom of a separate cluster of 10 keys between the keypad and the standard keys. The <BS> key is labeled "BACK SPACE" on the VT100-type keyboards. On the VT200 series keyboards, it is the <F12> key on the top row of "function keys" — the one just above the <+> key. The <LF> key is labeled "LINE FEED" on the VT100 type keyboards. On the VT200 series keyboards, it is the <F13> key just above the <DEL> key (which is labeled with an "X" in a pentagon). The <CTRL> key is located at the far left of the keyboard. (See Figs. 1-1 and 1-2.)

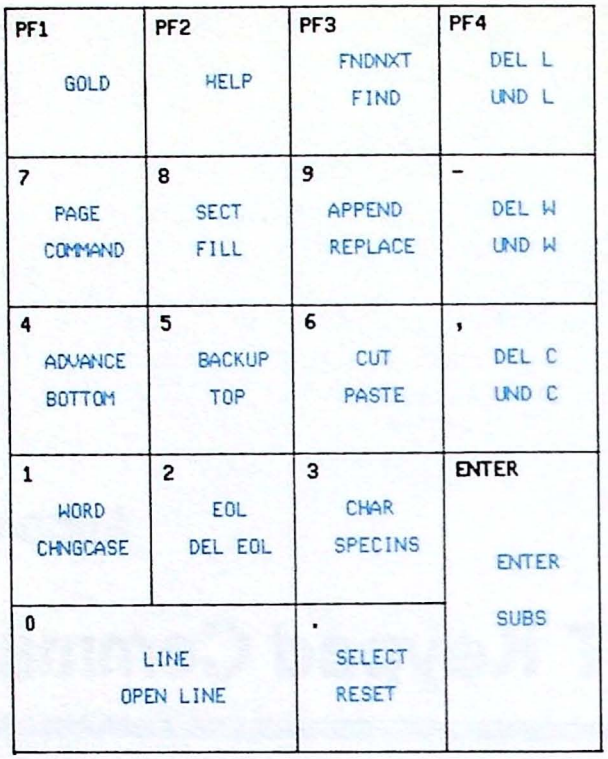

Fig. D-1. The keypad used in the Keypad Mode.

Just as in the other VAX modes, each command in the Keypad Mode is an ordinary English verb or verb phrase. But unlike the other modes in which the commands must be spelled out letter-by-letter, these commands are executed by pressing just one or two keys. In the listing that follows, the name of each command is given first. Directly after the name, the key combination which is to be pressed is given in angle brackets. For example, the heading "TOP <PF1/5>" means that the name of the command is TOP, and that the command is executed by pressing the <PF1> key followed by the <5> key (on the keypad). Combinations like <CTRL/R> which use the <CTRL> key require that it be held down while the second key is pressed, whereas combinations which use the <PF1> key only require that it be pressed before the second key.

ADVANCE <4>

Sets the current direction forward, toward the end of the file.

APPEND <9>

Deletes the Select Range and appends it to the Paste Buffer.

BEGINNING OF LINE <BS>

Moves the cursor to the beginning of the line.

BACKUP <5>

Sets the current direction backwards, toward the beginning of the file.

BOTTOM <PF1/4>

Moves the cursor to the end of the file.

CARRIAGE RETURN <CTRL/M>

The Carriage Return character (ASCII 13), often denoted by <CR>, is used in applications to move the print head of a printer to the beginning of the current line, as though "returning the carriage" of a typewriter.

CHARACTER <3>

Moves the cursor to the next character in the current direction.

CHANGE CASE <PF1/1>

Changes the case (uppercase to lowercase and vice versa) of all letters in the Select Range.

COMMAND <PF1/7>

Prompts the user to give an EDT command. Press <ENTER> to execute the command.

CUT <6>

Moves the Select Range from the current buffer into the Paste Buffer.

DEFINE KEY <CTRL/K>

Allows you to define a key to execute a sequence of commands automatically. After selecting the key combination (e.g., <PF1/G>) which you want to define, enter the commands which you want to execute automatically. Add a dot (.) at the end if you want the execution to be immediate. Then, enter the definition by pressing <ENTER> on the keypad. Use the INSERT Mode, delineated by I and <CTRL/Z>, if you want to include any text in your definition.

DELETE CHARACTER <.>

Deletes the current character (under the cursor).

DELETE LINE <PF4>

Deletes all characters from the current character to the beginning of the next line, including the end-of-line terminator. This causes the next line to be moved up, putting its first character at the current cursor position.

DELETE TO BEGINNING OF WORD <LF>

Deletes all characters in the current word which are to the left of the cursor.

DELETE TO END OF LINE <PF1/2>

Deletes all characters from the current character to the end of the line. It does not delete the end-of-line terminator, so the next line remains unaffected.

DELETE WORD <->

Deletes all characters from the current character to the beginning of the next word. This causes the next word to be moved to the left, putting its first character at the current cursor position.

ENTER <ENTER>

Used to execute the COMMAND, DEFINE KEY, and FIND functions.

END OF LINE <2>

Moves the cursor to the end of the line.

EXIT <CTRL/Z>

Exits from the Keypad Mode, returning to EDT Line Mode.

FILL <PF1/8>

Reformats the select range, fitting as many words as possible on each line without exceeding the screen width (the default value is 80).

FIND <PF1/PF3>

Prompts the user to enter a text string into the Search Buffer. Once entered, this string will be searched for in the current direction. Use the <ENTER> key to enter the string.

FIND NEXT <PF3>

After a text string has been entered into the Search Buffer (by means of the FIND command), the FIND NEXT command will continue the search in the current direction for the next occurrence of the same string.

FORM FEED <CTRL/L>

The Form Feed character (ASCII 12), denoted by <FF> on the screen, is used in many applications to mark the end of a page. Printer terminals will "feed the form," advancing the paper to the next page, when they encounter this character. The PAGE command will advance the cursor to the next <FF>.

HELP <PF2>

Displays a picture of the keypad, asks which key is to be described, and then displays a brief description of the function of that key.

LINE <0>

Moves the cursor to the beginning of the next line (in the current direction).

OPEN LINE <PF1/0>

Inserts an end-of-line terminator (i.e., a line feed) to the right of the cursor. This creates a new line for everything from the current character to the end of the line. The OPEN LINE command has the same effect as pressing <RET>, except that the cursor is not moved down to the beginning of the next line.

PAGE <7>

Moves the cursor to the beginning of the next page (in the current direction). The default page delimiter is the Form Feed character <FF> (ASCII 12), inserted by <CTRL/L>.

PASTE <PF1/6>

Inserts a copy of the Paste Buffer into the file at the current position.

REFRESH SCREEN <CTRL/R>

Removes spurious characters from screen.

REPEAT COMMAND <PF1>

When followed by a repetition factor (a positive integer, N, typed on the standard keyboard), this causes the next command to be repeated N times.

REPLACE <PF1/9>

Deletes the text in the Select Range and replaces it with the contents of the Paste Buffer.

RESET <PF1/.>

Cancels the current Select Range. It may also be used to cancel a partially-entered command, such as one being entered with the <PF1> key.

SECTION <8>

Moves the cursor 16 lines (one section) in the current direction.

SELECT <.>

Used to define the Select Range by marking one end. The Select Range consists of all characters between the position marked by the SELECT command and the current position (marked by the cursor). It is used with the APPEND, CHANGE CASE, CUT, REPLACE, and SUBSTITUTE commands (among others).

SPECIAL INSERTION <PF1/3>

Allows the insertion of special characters by means of their ASCII Code. (See Appendix K.) For example, the sequence <PF1>7<PF1/3> will insert the

ASCII character BELL (the character which produces a "beep" when displayed or printed), because 7 is the ASCII code for BELL. The code number must be typed on the standard keyboard.

SUBSTITUTE <PF1/ENTER>

Used in conjunction with the FIND NEXT command. The sequence <PF3/PF1/ENTER> will locate the next occurrence (in the current direction) of the string in the Search Buffer and replace it with the string in the Paste Buffer.

TOP <PF1/5>

Moves the cursor to the first character of the file.

UNDELETE CHARACTER <PF1/,>

Inserts at the current position the last character that was deleted.

UNDELETE LINE <PF1/PF4>

Inserts at the current position the last line that was deleted.

UNDELETE WORD <PF1/->

Inserts at the current position the last word that was deleted.

WORD <1>

Moves the cursor (in the current direction) to the beginning of the next word.

Appendix E

Minitab Commands

Statistics is that analysis of data which enables us to learn by experience.

Andrew Gleason

This appendix contains brief descriptions of the most commonly used Minitab commands. It follows the notation used in the *Minitab Reference Manual* by Thomas A. Ryan, Jr., Brian L. Joiner, and Barbara F. Ryan: K stands for a constant label (e.g., K5) or a range of constant labels (e.g., K1 – K3), C stands for a column label (e.g., C2) or a range of column labels, and E stands for either C or K. Anything enclosed in square brackets [] means that it is optional; for example, *SAVE [in 'filespec']* stands for both "SAVE" and "SAVE in 'filespec'". Anything enclosed in curly brackets {} means that it is both optional and repeatable; for example, *READ 'filespec' into C {,C}* stands for "READ 'filespec' into C", "READ 'filespec' into C, C" and "READ 'filespec' into C, C, C", etc. Capitalized words are commands, and the symbols and words printed in color are required, the rest is optional (and is ignored by the system). The word "filespec" stands for a file specification, such as SCORES.MTB, and the word "string" stands for any character string. The word "expression" stands for any algebraic combination of constants and columns, for example: K3 + 7.19/C2**4 – SQRT (ABSO(MEAN(C1))).

ABSOLUTE VALUE OF E INTO C

Puts the absolute value of E into C.

Example: ABSO C6 INTO C2

ADD E TO E AND PUT INTO E

Puts the sum of the first two into C or K.

Example: ADD C1 to C2 INTO C3

CENTER C {,C} INTO C {,C}

Puts the Z-score of the nth C into the (2n)th C.

Example: CENTER C1, C2, C3 INTO C4, C5, C6

CHOOSE VALUES K {THROUGH K} in C {,C} PUT INTO C{,C}

Selects rows which contain values in a given range.

Example: CHOOSE 35 TO 75 IN C1-C3, INTO C11-C13

CODE K {TO K} IN C TO K, AND PUT INTO C

Copies the column, changing the indicated values.

Example: CODE (0) TO 10 IN C2 INTO C7

COPY C {,C} INTO C {,C}

Copies the columns indicated.

Example: COPY C5, C2-C4 INTO C12-C15

CORRELATE C, C, {,C}

Prints the correlation coefficients

Example: CORR C4 AGAINST C3

COUNT LENGTH OF C {AND PUT INTO E}

Displays the length of the first column, and (if requested) also puts it in E.

Example: COUNT C4 INTO K2

DEFINE K INTO C

Puts the value of the constant into each row of C.

Example: DEFINE 3.14 INTO C7

DESCRIBE C {,C}

Prints COUNT, MEAN, MEDIAN, STDEV, MAX, and MIN

Example: DESC C1-C3, C8

DIVIDE E BY E AND PUT INTO E

Puts the quotient of the first two into C or K.

Example: DIVIDE C1 BY C2 INTO C3

END STORING MINITAB COMMANDS IN MACRO

Used to end creation of macro (see STORE).

Example: END MACRO

ERASE E {,E}

Erases all constants and columns specified.

Example: ERASE C21-C29, K5-K8, C9

EXECUTE COMMANDS [STORED IN '*filespec*'] [K TIMES]

Executes the macro (see STORE).

Example: EXECUTE MACRO 5 TIMES

EXPONENTIAL OF E INTO E

Puts e to the power E into E.

Example: EXPO K9 INTO C3

GENERATE THE FIRST K INTEGERS INTO C

Puts 1, 2, 3, . . , K into the first K rows of C.

Example: GENERATE 16 INTO C9

HELP

Prints information about Minitab.

Example: HELP

HISTOGRAM C {,C} [WITH FIRST MIDPOINT K AND WIDTH K]

Prints a histogram for each column specified; the optional constants allow you to set the midpoints and interval widths.

Example: HIST C1 MIDPOINT 450 WIDTH 25

INFORMATION {ON C}

Shows all constants and columns used in the workspace.

Example: INFO

INSERT [FROM '*filespec*'] BETWEEN ROWS K AND K OF C {,C}

Reads data into the specified rows of the given columns; the two column numbers must be consecutive.

Example: INSERT BETWEEN 5 AND 6 OF C2, C4-C6

JOIN E TO E {TO E}, and PUT INTO C

Appends each constant or column to the next, and puts the final concatenated column into C.

Example: JOIN C1, C2, 355, K1, C3, INTO C16

LET E = *expression*

Puts the value of the expression into E.

Example: LET C4 = C1(COUNT(C2)) − STDEV(C3)**2

LOGE OF E INTO E

Puts the natural logarithm (base e) into the column or constant specified.

Example: LOGE OF K3 INTO C7

LOGTEN OF E INTO E

Puts the common logarithm (base 10) into C or K.

Example: LOGTEN OF C3 INTO C7

MAXIMUM OF C {AND PUT INTO E}

Displays the maximum of the first column, and (if requested) also puts it in E.

Example: MAXIMUM OF C4 INTO K2

MINIMUM OF C {AND PUT INTO E}

Displays the minimum of the first column, and (if requested) also puts it in E.

Example: MINIMUM OF C4 INTO K2

MEAN OF C {AND PUT INTO E}

Displays the mean average of the first column, and (if requested) also puts it in E.

Example: MEAN OF C4 INTO K2

MEDIAN OF C {AND PUT INTO E}

Displays the median of the first column, and (if requested) also puts it in E.

Example: MEDIAN OF C4 INTO K2

MULTIPLY E BY E AND PUT INTO E

Puts the product of the first two into C or K.

Example: MULTIPLY C1 BY K1 INTO C3

NAME FOR C IS *'string'* {, FOR C IS *'string'* }

Assigns the string as a name for the column.

Example: NAME C1 'AGE', C2 'Height'

OMIT VALUES K {THROUGH K} IN C {,C} AND PUT INTO C{,C}

Removes rows which contain values in the given range.

Example: OMIT 350 THROUGH 750 IN C1-C3, INTO C11-C13

ORDER C {,C} AND PUT INTO C {,C}

Sorts each column independently.

Example: ORDER C1-C3 INTO C21-C23

OUTFILE [INTO *'filespec'*]

Redirects all output, either to the terminal or to a file.

Example: OUTFILE 'MTBOUTPUT.TXT'

PICK ROWS K THROUGH K OF C, AND PUT INTO C

Copies the selected row range into C.

Example: PICK ROWS 6 THROUGH 12 OF C2, INTO C7

PLOT Y-COORDINATES IN C AGAINST X-COORDINATES IN C

Plots one point for each row.

Example: PLOT C2 AGAINST C7

PRINT C {,C}

Displays the columns listed.

Example: PRINT C2

RAISE E TO E AND PUT INTO E

Raises the first to the power of the second, and puts the result into C or K.

Example: RAISE C1 TO 4 INTO C3

RANK C AND PUT RANKS INTO C

Saves the rank numbers of the given column.

Example: RANK C5 INTO C45

READ [FROM *'filespec'*] THE FOLLOWING DATA INTO C {,C}

Reads data into the workspace, one line per row.

Example: READ 'SCORES.DAT' into C1-C3, C5

REGRESS C ON K PREDICTORS C {,C}

Computes the regression equation with K inputs.

Example: REGRESS C1 ON 7 PREDICTORS C61-C67

RESTART

Clears the workspace.

Example: RESTART

RETRIEVE FROM *'filespec'*

Reads data into the workspace; external file must have been created by SAVE.

Example: RETRIEVE 'SYS$MINITAB:GRADES'

ROUND E INTO C
Puts the nearest integer of E into C.
Example: ROUND C7 INTO C2

SAMPLE K FROM C AND PUT INTO C
Selects K rows from the first column at random and puts those values into the second column.
Example: SAMPLE 32 FROM C1 INTO C2

SAVE [IN *'filespec'*]
Copies the entire worksheet into the file; the default file is MINITAB.MTW.
Example: SAVE 'GRADES.MTB'

SET [FROM *'filespec'*] THE FOLLOWING DATA INTO C
Reads all the data into one column in the workspace.
Example: SET 'CENSUS.DAT' INTO C4

SIGN OF E INTO E
Puts the sign of E (1, 0, or −1) into E.
Example: SIGN OF C1 INTO C3

SORT C {, CARRYING C} AND PUT INTO C {,C}
Sorts the first column, and reorders columns specified using the ordering from that sort.
Example: SORT C1 CARRYING C2-C3 INTO C31-C33

SQRT OF E INTO E
Puts the nonnegative square root of E into E.
Example: SQRT OF C5 INTO C4

STDEV OF C {AND PUT INTO E}
Displays the standard deviation of the first column, and (if requested) also puts it in E.
Example: STDEV OF C4 INTO K2

STOP
Returns to DCL Mode.
Example: STOP

STORE THE FOLLOWING MINITAB COMMANDS [in *'filespec'*]
Creates one macro (see END and EXECUTE).
Example: STORE IN 'MTBMACRO.TXT'

SUBSTITUTE K INTO ROW K OF C

Puts the value of the first constant into the row of C indexed by the second constant.

Example: SUBS K3 INTO ROW 5 OF C4

SUBTRACT E FROM E AND PUT INTO E

Puts the difference of the two values into C or K.

Example: SUBTRACT 7.23 FROM C1 INTO C3

SUM C {AND PUT INTO E}

Displays the sum of the first column, and (if requested) also puts it in E.

Example: SUM C4 INTO K2

WRITE TO *'filespec'* C {,C}

Copies the columns specified into the file.

Example: WRITE TO 'GRADES.DAT' C1-C6

Appendix F

BASIC Commands

We were not born to sue, but to command.

Shakespeare (Richard II, I.i.182)

This appendix lists most of the commands which you can use in the VAX BASIC Mode. As in the other appendixes, the minimal abbreviation for each command is indicated with colored print.

You can get on-line information about these commands with the HELP command in BASIC Mode. For example, type HELP COMMAND LIST to get information on the LIST command.

APPEND *filename*

Copies the file with specified filename and type BAS into the buffer at the end of its current contents. If any line numbers are duplicated, the lines of the specified file replace those in the buffer.

Example: APP PROGRAM1

COMPILE

Compiles the current BASIC program in the buffer. This translates the program into machine language. Then, after the LOAD command is executed, the program can be RUN without any more translating. Using compiled BASIC this way is more efficient than the usual interpreted BASIC.

Example: COM

CONTINUE

After being interrupted with a <CTRL/Y>, a <CTRL/C>, or a STOP statement, the system will return to where it left off if the CONTINUE command is the next command executed.

Example: CON

DELETE *number-list*

Removes the specified lines from the current program.

Example: DEL 120, 150-180

EDIT *number/string/string/*

If a line number and substitute strings are specified, then the EDIT command replaces the first string with the second string in the line with the specified line number. Otherwise, the EDT editor is invoked.

Example: EDI 230/INPUT/PRINT/

EXIT

If the current program has been changed and not saved, then a warning message will be displayed. Otherwise, control returns to the DCL Mode.

Example: EXI

LIST *number-list*

Displays the specified lines in the current program. If no line numbers are specified, then the entire program is displayed.

Example: LIS 220-260, 397, 512

LISTNH

Has the same effect as the LIST command, except that the header is not displayed. ("NH" is for "No Header.")

Example: LISTN

LOAD

Loads the object module produced by the COMPILE command, so that the compiled BASIC program can be RUN.

Example: LOA

NEW *filename*

Empties the buffer. If a filename is specified, then that name will be given to the program written next in the buffer.

Example: NEW PROGRAM3

OLD *filename*

Empties the buffer and then copies into it the BASIC program that was saved with the filename specified.

Example: OLD PROGRAM2

RENAME *filename*

Changes the name of the current BASIC program to the filename specified.

Example: REN PROGRAM5

RUN

Runs the current BASIC program.

Example: RUN

RUNNH

Runs the current BASIC program without printing a header first.

Example: RUNN

SAVE *filename*

Copies the current BASIC program into a file. The filetype is BAS. If no filename is specified, then the last one used is given to the new file. If a BAS file already exists with that filename, then a new version is created.

Example: SAV

SCRATCH

Clears the buffer and resets the current program name to NONAME. The current program will be lost unless it is saved first.

Example: SCR

SEQUENCE *number, number*

Enters the SEQUENCE Mode. In this mode, line numbers are generated automatically. If no numbers are specified, then the numbers increment by 10 and begin after the last line number in the buffer (or with 100 if the buffer is empty). Otherwise, the numbering begins with the first number specified and uses the second number as the increment.

Example: SEQ 2000, 5

UNSAVE *filename*

Deletes the file with the specified filename and filetype BAS.

Example: UNS PROGRAM1

Appendix G

UNIX Shell Commands

"When I use a word," Humpty Dumpty said in a rather scornful tone, "it means just what I choose it to mean — neither more nor less." "The question is," said Humpty Dumpty, "which is to be the master — that's all."

Lewis Carroll (Through the Looking-Glass, Ch. 7

The UNIX operating system actually consists of many programs. The one program that is always running and which controls all the others is called the *kernel.* Another unique program serves to interpret commands to the kernel. This program is called the *UNIX shell,* and the system commands are called *shell commands.* In contrast, the VAX/VMS system commands are called *DCL commands.*

This appendix lists some of the more common shell commands which are available on the VAX emulator EUNICE. The origin of each command name is indicated first, followed by options and a brief description of what the command does. If there is a VMS DCL equivalent command, it is listed with its UNIX equivalent. Finally, examples are given. The percent sign is the EUNICE prompt.

cal
For "calendar"
Displays the daily calendar of the month or year specified.
Examples: % cal 1987
% cal 11 1965

cat
For "concatenate-and-print"
Options: -n = number the lines
-v = include nonprinting characters
Displays the contents of one or more files. See: 1pr, pr.
DCL: TYPE
Examples: % cat Russell
% cat Mao* Pascal *1?

cd
For "change directory"
Changes the current (default) directory.
DCL: SET DEFAULT
Examples: % cd data
% cd /
% cd /usr/dict
% cd ../../data

chmod
For "change mode"
Changes the protection code for the file(s) specified.
DCL: SET PROTECTION
Examples: % chmod 700 test.dat
% chmod 636 s*

cmp
For "compare"
Compares two files and reports where they first differ. If the files are identical, no report is given. See: comm, diff.
DCL: DIFFERENCES
Example: cmp Shakespeare Thought

comm
For "common lines"
Options: -1 = suppress column 1
-2 = suppress column 2
-3 = suppress column 3
Determines which lines are common to the two files specified. The lines of both files are displayed. Those which are only in the first file are printed in column 1, those which are only in the second file are printed in column 2, and those which are common to both are printed in column 3. The options cause the column specified to be omitted from the display. Thus, the option -13 would cause only column 2 to be printed; i.e., only the lines which are in the second file but not in the first would be displayed. See: cmp, diff, sort.
DCL: DIFFERENCES
Example: comm -1 Shakespeare Thought

cp

For "copy"

Makes a copy of the file specified and gives it the name specified. If a directory is specified with no filename, then the new copy in the new directory is given the same name as the original.

DCL: COPY

Examples: % cp Mao_Tse-Tung Mao
% cp data ../temp/

crypt

For "encryption-decryption"

Encrypts text files into encrypted files and decrypts encrypted files back into readable text files. To "encrypt" means to translate ordinary text into coded text which is unreadable by humans, and "decrypt" means to reverse that process. The UNIX shell will ask you to enter a keyword when you encrypt the file. The same keyword must be supplied to decrypt it. The encryption/decryption process is time-consuming.

Examples: % crypt <report >secret
% crypt <secret >report

date

For "date-time"

Displays the date and time.

DCL: SHOW TIME

Example: % date

diff

For "differences"

Options: -b = ignore trailing blanks
-e = display ed script

Determines what makes one file different from another. The -e option creates a list of ed commands which would transform the first file into the second. See: cmp, comm.

DCL: DIFFERENCES

Examples: % diff topten vowels
% diff -e topten vowels

ed

For "text editor"

Invokes the UNIX standard line editor.

DCL: EDIT

Examples: % ed
% ed memo

file

For "filetype"

Determines the type of the file(s) specified, such as data, text, empty, program, ascii, and directory.

Example: % file *.tmp

grep
For "global/regular-expression/print"
Options: -c = report only the number of lines found
-l = list files with matching lines
-n = print the filenames and line numbers
-v = find all lines that do not contain the given string
-y = make no distinction between uppercase and lowercase
Searches for the specified character string in the specified file(s).
DCL: SEARCH
Examples: % grep " the " Russell
% grep -nyv "thought" S*

learn
Invokes a computer-assisted learning course. Unix will teach you about files, its editor, and other essentials. Simply follow the on-line directions.
Example: % learn

lpr
For "line-printer"
Queues the files specified to the system line printer.
DCL: PRINT
Example: % lpr test

ls
For "list"
Options: -d = list only directory files (subdirectories)
-l = include info on protection, size, owner, etc.
-m = list files in stream mode
-r = list in reverse order
-s = give size in blocks
-t = list in chronological instead of alphabetical order
-u = use last access time instead of last modification time
-F = mark directory files with a slash (/)
Lists the contents of directories specified.
DCL: DIRECTORY
Examples: % ls
% ls /usr/skel
% ls -surt ../*.dat
% ls -d tmp/test?

mail
Allows you to send mail to other users and to read your own mail.
DCL: MAIL
Example: mail gorbachev

man
For "manual"
Option: -w = show subdirectory paths of submanuals
Displays the chapter(s) and section(s) of the UNIX User's Manual, unless the -w option is used.
DCL: HELP
Example: % man

mkdir
For "make-directory"
Creates the subdirectory specified.
DCL: CREATE/DIRECTORY
Example: %mkdir old_stuff

mv
For "move"
Changes the filename of the file specified to the name specified.
DCL: RENAME
Example: % mv speech nonsense

pr
For "print"
Options: -lN = use number N of lines per page specified
-wN = use number of columns per page specified
Displays the file(s) specified in a format of pages with headers. The default page size is 66 lines and 72 columns. See: cat, lpr.
DCL: TYPE
Examples: % pr memo
% pr -w64 list?

pwd
For "path-of-working-directory"
Displays the complete directory path from the root directory to the current subdirectory.
DCL: SHOW DEFAULT
Example: % pwd

rm
For "remove-files"
Options: -i = ask for verification for each file to be deleted
-r = ask for verification for each directory file to be deleted
Deletes file(s) specified. If the -r option is used alone on a directory file, that subdirectory and its entire contents will be deleted automatically. If the -ir option is used on a directory file, then you will be queried on the deletion of every file in the subdirectory before the subdirectory is deleted. See: rmdir.
DCL: DELETE
Examples: % rm Thought
% rm -ir old_stuff

rmdir
For "remove-subdirectory"
Used to delete the directory file of an empty subdirectory. See: rm.
DCL: DELETE
Example: % rmdir old_stuff

sort
Options: -b = ignore leading blanks
-d = ignore all but letters, digits, and blanks
-f = ignore case distinction
-i = ignore characters outside the ASCII range 32-126
-m = merge files
-n = sort leading numeric strings numerically
-r = reverse order
-tX = use character X as a line separator
-u = delete duplicate lines
Displays the lines of the specified file(s) in sorted order.
DCL: SORT
Examples: % sort list?
% sort -fir Russell

vms
For "VAX/VMS"
Allows the execution of DCL commands from within the UNIX emulator EUNICE.
Example: % vms SHOW QUOTA

wc
For "word-count"
Options: -c = just count characters
-l = just count lines
-w = just count words
Displays the number of lines, words, and characters of the file(s) specified.
Examples: % wc Thought
% wc -wl S*

who
Displays a list of users currently working with UNIX. See: whoamI.
DCL: SHOW USERS
Example: % who

whoami
For "who-am-I"
Displays your username. See: who.
Example: % whoami

write
Sends an immediate message to another user.
DCL: PHONE
Example: % write gorbachev

Appendix H

UNIX Text Editing Commands

Kindly enter them into your note-book. And, in order to refer to them conveniently, let's call them A, B, and Z.

Lewis Carroll (What the Tortoise Said to Achilles)

This appendix lists the UNIX ed commands. As described in Chapter 11, many of these commands accept an optional reference to a line number or to a range of lines. A single line number is indicated by a single number sign (#). A range of lines is indicated by two number signs separated by a comma (#,#). For example, the listing for the print command is #,#p, meaning that the p command may stand alone, may be preceded by a single line number, or may be preceded by a line number range.

A line number may be specified by an integer, a dollar sign (referring to the last line in the buffer), a dot (referring to the current line), a plus sign (referring to the line below the current line), a minus sign (referring to the line above the current line), a search string in the form /string/ (referring to the next line containing the string), or any one of these symbols followed by a displacement, such as +3 or −2.

Most commands can be followed by p, which would print the last line affected by the command. This last line becomes the current line.

#a
For "add"
Invokes the insert mode, allowing lines of text to be typed into the editing buffer. The new lines are inserted below the current line. The mode is terminated by typing the dot (.) by itself on a separate line.
Example: 5a

#,#c
For "change"
Invokes the insert mode, allowing lines of text to be typed into the editing buffer, with the new lines replacing those in the specified range. The mode is terminated by typing the dot (.) by itself on a separate line.
Example: 5,8c

#,#d
For "delete"
Deletes the lines specified.
Example: 4,/stand./d

e filespec
For "edit"
Replaces the current contents of the editing buffer and then reads in the file specified. This is equivalent to a delete followed by a read.
Example: e poem

f
Displays the current filespec.
Example: f

#i
For "insert"
Invokes the insert mode, allowing lines of text to be typed into the editing buffer. The new lines are inserted above the current line. If the editing buffer is empty, then the add command must be used to insert new lines. The mode is terminated by typing the dot (.) by itself on a separate line.
Example: 5i

#,#j
For "join"
Combines the lines in the range specified.
Example: .+1,$−2j

#,#m#
For "move"
Moves the specified range of lines to the position directly below the target line specified. This is equivalent to a transfer followed by a delete.
Example: $−2,$m.+3

#,#p
For "print"
Displays the range of lines specified.
Example: 1,$p

q
For "quit"
Terminates the editing session, returning you to UNIX mode. The contents of the file buffer are lost when this is done.
Example: q

#r filespec
For "read"
Copies the contents of the file specified into the editing buffer immediately below the line specified. This is the reverse of the write command.
Example: 5r paragraph-12.7

#,#s/string/string/
For "substitute"
Option: g = replace every occurrence in the specified range
Replaces the first string with the second string in the specified range. If the option g is appended, then every occurrence will be replaced; otherwise, only the first occurrence is replaced.
Example: 1,.s/they/he or she/g

#,#t#
For "transfer"
Copies the specified range of lines to the position directly below the target line specified.
Example: $-5,$t.

#,#w filespec
For "write"
Copies the contents of the range specified into the file. This is the reverse of the read command.
Example: w paragraph

=
Displays the number of lines in the buffer.

.
Displays the current line.

<RET>
Resets the current line to the line directly below the current line (i.e., increments the current line pointer) and then displays this new current line. This is equivalent to +p.

Appendix I

MAIL Commands

The medium is the message.

Marshall McLuhan (Understanding the Media, 1964)

The VAX/VMS Mail Utility (MAIL) allows you to send messages and files to other users, whether or not they are logged in to the system at the time. You can also file, forward, delete, print, and reply to messages which other users send to you. The messages you receive are stored in your mail file, MAIL.MAI.

The DCL command MAIL invokes the MAIL Utility, moving you into the MAIL Mode. The MAIL command EXIT will return you to the DCL Mode. (See Appendix A.)

All the messages in your mail file are categorized in MAIL *folders.* Three folders exist automatically; they are called MAIL, NEWMAIL, and WASTEBASKET. All other folders must be created by you. You can create new folders by means of the COPY, FILE, and MOVE commands. You can reset your current folder by means of the DIRECTORY command.

When you receive a new MAIL message, its arrival is announced on your screen, and the message is automatically deposited into your NEWMAIL folder. When you read the new message for the first time, it is automatically transferred to your MAIL folder. When you DELETE a message, it is transferred to your WASTEBASKET folder. Each time you EXIT from the MAIL utility, all the messages in your WASTEBASKET folder are automatically deleted; that is, your WASTEBASKET is emptied.

Many of the MAIL commands are with reference to the *current message.* This is the last message read by you. You can set the current message by typing its message number as defined by the directory of its file folder.

The following commands may be entered at the MAIL> prompt. As with commands in other modes, these commands may be abbreviated to their first few letters. The colored letters are the minimal abbreviations. The word "filespec" stands for any file specification, "username" stands for any username, "N" stands for any message number (a positive integer), and the word "folder" stands for any MAIL folder name.

Many of the MAIL commands allow optional qualifiers, such as /EDIT, /EXTRACT, /LAST, /NOHEADER, and /SELF. The /EDIT qualifier allows you to use the EDT editor to edit your message. The /EXTRACT qualifier allows you to edit the current message which was sent to you into a new message to be sent by you. The /LAST qualifier does the same thing as the /EXTRACT qualifier, except that it uses the last message that you sent as text for the reply. The /NOHEADER qualifier removes the MAIL header from the message, leaving only the message itself. The /SELF qualifier causes a copy of your message to be sent to yourself.

ANSWER/ED/EX/L/S/*filespec*

Sends a message back to the sender of the current message. This command is equivalent to the REPLY command.

BACK

Backs up to the previous message.

COPY *folder*

Copies the current message to the specified folder. If the folder does not exist, then it will be created. This has the same effect as the FILE and MOVE commands, except that a copy of the message is kept in its current folder.

DEFINE/KEY/TERMINATE

Allows you to define keyboard and keypad keys to execute MAIL commands. This works the same way as the DEFINE/KEY command in DCL. (See Appendix B.)

DELETE

Moves the current message to the WASTEBASKET folder.

DIRECTORY/BEFORE/FOLDER/FULL/NEW/SINCE *folder*

Lists the messages in your current folder. The name of your current folder is displayed in the upper right corner. If a folder is specified, then that becomes your current folder. The qualifier /BEFORE=2-AUG-1986 would include only those messages which were sent before that date. The /SINCE qualifier works in the opposite way. The /FOLDER qualifier displays only the existing folder

names. The /FULL qualifier produces the number of lines in each message, and whether you have replied. The /NEW qualifier lists only your new (unread) messages.

ERASE

Clears the screen.

EXIT

Exits the MAIL Mode, returning to DCL Mode. This is equivalent to PURGE followed by QUIT.

EXTRACT/APPEND/NOHEADER *filespec*

Creates a sequential (ASCII text) file and copies the current message into it. The /APPEND qualifier causes the message to be appended to the file specified instead of creating a new version. The /NOHEADER qualifier removes the MAIL heading before copying the message to the file.

FILE *folder*

Moves the current message to the specified folder. If the folder does not exist, then it will be created. This has the same effect as the MOVE command.

FIRST

Displays the first message in the current folder.

FORWARD/EDIT/NOHEADER *username*

Sends the current message to the user named. If you attach the /EDIT qualifier, then you will be able to edit your message.

HELP

Displays information on the MAIL utility.

LAST

Displays the last message.

MAIL/EDIT/LAST/SELF *filespec*

If a file is specified, then that file is sent to the user specified at the "To:" prompt. Otherwise, this command is the same as the SEND command.

MOVE *folder*

Moves the current message to the specified folder. If the folder does not exist, then it will be created. This has the same effect as the FILE command.

NEXT

Advances to your next mail message.

PURGE

Deletes all the messages in the WASTEBASKET folder.

QUIT

Exits from the MAIL Mode without emptying the WASTEBASKET folder.

READ/BEFORE/EDIT/SINCE *folder N*

Displays message number N in the folder specified (or the current folder if none is specified). If no message number is specified, then the next most recent (after the current message) is displayed. The default current folder is NEWMAIL. If the current folder is empty, then the MAIL folder becomes the current folder.

REPLY/EDIT/EXTRACT/SELF *filespec*

Sends a reply message to the sender of the current message. If a file is specified, then it is sent instead of a message.

SEARCH *string*

Searches all the messages in the current folder for the character string specified. If it is found, the message is displayed and it becomes the current message.

SEND/EDIT/LAST/SELF *filespec*

If a file is specified, then it is sent; otherwise, a message is sent.

SET

Used to enable or disable any of the following switches: AUTO_PURGE, COPY_SELF, PERSONAL_NAME.

SHOW

The SHOW command is used to see which switches have been enabled by the SET command. SHOW ALL will give you complete information.

Appendix J

The ASCII Code

Many persons who are not conversant with mathematical studies imagine that because the business of Babbage's Analytical Engine is to give its results in numerical notation, the nature of its processes must consequently be arithmetical and numerical rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols.

Ada Augusta (1844)

The acronym "ASCII" stands for "American Standard Code for Information Interchange." This code is the most widely used method for assigning numerical values to text characters. It has been standardized and certified by the American National Standards Institute (ANSI), which includes representatives from organizations such as the Air Transport Association (ATA), the American Library Association (ALA), the Association for Computing Machinery (ACM), the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE), and the U.S. Department of Defense (DoD), in addition to computer manufacturers, such as Digital Equipment Corporation (DEC) and International Business Machines Corporation (IBM).

The Standard ASCII Character Set consists of the 128 symbols given in Fig. J-1. Here, the numeric code is given in decimal form (base 10).

The "Keyboard Key" column gives the key(s) on the VT100 keyboard (refer to Fig. 1-1) that transmits the given character.

ASCII	Symbol	Keyboard Key	Name
0	NUL	<CTRL/@>	Null
1	SOH	<CTRL/A>	Start of Heading
2	STX	<CTRL/B>	Start of Text
3	ETX	<CTRL/C>	End of Text (Cancel)
4	EOT	<CTRL/D>	End of Transmission
5	ENQ	<CTRL/E>	Enquiry
6	ACK	<CTRL/F>	Acknowledge
7	BEL	<CTRL/G>	Bell
8	BS	<BACK SPACE>	Backspace
9	HT	<TAB>	Horizontal Tabulation
10	LF	<LINE FEED>	Line Feed
11	VT	<CTRL/K>	Vertical Tabulation
12	FF	<CTRL/L>	Form Feed (New Page)
13	CR	<CTRL/M>	Carriage Return
14	SO	<CTRL/N>	Shift Out
15	SI	<CTRL/O>	Shift In
16	DLE	<CTRL/P>	Data Link Escape
17	DC1	<CTRL/Q>	Device Control 1
18	DC2	<CTRL/R>	Device Control 2
19	DC3	<CTRL/S>	Device Control 3
20	DC4	<CTRL/T>	Device Control 4
21	NAK	<CTRL/U>	Negative Acknowledgement
22	SYN	<CTRL/V>	Sychronous Idle
23	ETB	<CTRL/W>	End of Transmission Block
24	CAN	<CTRL/X>	Cancel
25	EM	<CTRL/Y>	End of Medium (Abort)
26	SUB	<CTRL/Z>	Substitute (Exit)
27	ESC	<ESC>	Escape
28	FS	<CTRL/\>	File Separator
29	GS	<CTRL/]>	Group Separator
30	RS	<CTRL/^>	Record Separator
31	US	<CTRL/_>	Unit Separator
32	(space)	space Bar	Space (Blank)
33	!	<SHFT/1>	Exclamation Point
34	"	<SHFT/'>	Quotation Marks (Diaeresis)
35	#	<SHFT/3>	Number Sign
36	$	<SHFT/4>	Dollar Sign
37	%	<SHFT/5>	Percent Sign
38	&	<SHFT/7>	Ampersand
39	'	<'>	Apostrophe (Closing Single quotation Mark)
40	(	<SHFT/9>	Opening Parenthesis
41	)	<SHFT/10>	Closing Parenthesis
42	*	<SHFT/8>	Asterisk (Star)
43	+	<SHFT/=>	Plus
44	,	<,>	Comma
45	-	<->	Hyphen (Minus)
46	.	<.>	Period (Decimal Point, Dot)
47	/	</>	Slant (Divide)
48	0	<0>	Digit Zero
49	1	<1>	Digit One
50	2	<2>	Digit Two
51	3	<3>	Digit Three

52	4	⟨4⟩	Digit Four
53	5	⟨5⟩	Digit Five
54	6	⟨6⟩	Digit Six
55	7	⟨7⟩	Digit Seven
56	8	⟨8⟩	Digit Eight
57	9	⟨9⟩	Digit Nine
58	:	⟨SHFT/;⟩	Colon
59	;	⟨;⟩	Semicolon
60	<	⟨,⟩	Less Than
61	=	⟨=⟩	Equals
62	>	⟨.⟩	Greater Than
63	?	⟨SHFT//⟩	Question Mark
64	@	⟨SHFT/2⟩	Commercial At
65	A	⟨SHFT/A⟩	Uppercase Letter A
66	B	⟨SHFT/B⟩	Uppercase Letter B
67	C	⟨SHFT/C⟩	Uppercase Letter C
68	D	⟨SHFT/D⟩	Uppercase Letter D
69	E	⟨SHFT/E⟩	Uppercase Letter E
70	F	⟨SHFT/F⟩	Uppercase Letter F
71	G	⟨SHFT/G⟩	Uppercase Letter G
72	H	⟨SHFT/H⟩	Uppercase Letter H
73	I	⟨SHFT/I⟩	Uppercase Letter I
74	J	⟨SHFT/J⟩	Uppercase Letter J
75	K	⟨SHFT/K⟩	Uppercase Letter K
76	L	⟨SHFT/L⟩	Uppercase Letter L
77	M	⟨SHFT/M⟩	Uppercase Letter M
78	N	⟨SHFT/N⟩	Uppercase Letter N
79	O	⟨SHFT/O⟩	Uppercase Letter O
80	P	⟨SHFT/P⟩	Uppercase Letter P
81	Q	⟨SHFT/Q⟩	Uppercase Letter Q
82	R	⟨SHFT/R⟩	Uppercase Letter E
83	S	⟨SHFT/S⟩	Uppercase Letter S
84	T	⟨SHFT/T⟩	Uppercase Letter T
85	U	⟨SHFT/U⟩	Uppercase Letter U
86	V	⟨SHFT/V⟩	Uppercase Letter V
87	W	⟨SHFT/W⟩	Uppercase Letter W
88	X	⟨SHFT/X⟩	Uppercase Letter X
89	Y	⟨SHFT/Y⟩	Uppercase Letter Y
90	Z	⟨SHFT/Z⟩	Uppercase Letter Z
91	[	⟨[⟩	Opening Bracket
92	\	⟨\⟩	Reverse Slant
93	]	⟨]⟩	Closing Bracket
94	^	⟨SHFT/6⟩	Circumflex
95	_	⟨SHFT/-⟩	Underline
96	`	⟨`⟩	Grave Accent
97	a	⟨A⟩	Lowercase Letter a
98	b	⟨B⟩	Lowercase Letter b
99	c	⟨C⟩	Lowercase Letter c
100	d	⟨D⟩	Lowercase Letter d
101	e	⟨E⟩	Lowercase Letter e
102	f	⟨F⟩	Lowercase Letter f
103	g	⟨G⟩	Lowercase Letter g
104	h	⟨H⟩	Lowercase Letter h
105	i	⟨I⟩	Lowercase Letter i
106	j	⟨J⟩	Lowercase Letter j
107	k	⟨K⟩	Lowercase Letter k
108	l	⟨L⟩	Lowercase Letter l
109	m	⟨M⟩	Lowercase Letter m
110	n	⟨N⟩	Lowercase Letter n
111	o	⟨O⟩	Lowercase Letter o
112	p	⟨P⟩	Lowercase Letter p

113	q	<Q>	Lowercase Letter q
114	r	<R>	Lowercase Letter r
115	s	<S>	Lowercase Letter s
116	t	<T>	Lowercase Letter t
117	u	<U>	Lowercase Letter u
118	v	<V>	Lowercase Letter v
119	w	<W>	Lowercase Letter w
120	x	<X>	Lowercase Letter x
121	y	<Y>	Lowercase Letter y
122	z	<Z>	Lowercase Letter z
123	{	<SHFT/[>	Opening Brace (Left Curly Bracket)
124	|	<SHFT/\>	Vertical Line (Pipe)
125	}	<SHFT/]>	Closing Brace (Right Curly Bracket)
126	~	<SHFT/`>	Tilde
127	DEL	<DELETE>	Delete (Erase)

Fig. J-1. The ASCII Code.

Glossary

The beginning of wisdom is the definition of terms.

Socrates

abort — To interrupt an activity and return control to the previous mode or environment. In VAX/VMS, pressing <CTRL/C> will abort the execution of a program.

account — Each user of the VAX has an account, which contains the user's directory and all files. The account is identified by a username.

Ada — A modern high-level programming language developed for the U. S. Department of Defense. It is named after Ada Augusta, Countess of Lovelace, who was the daughter of the nineteenth-century English poet Lord Byron and is believed to be the first person to have written a computer program. The language Ada, incorporating the best features of two other languages, PL/1 and Modula-2, is well-suited for large-scale software systems.

address — A positive integer used by the operating system of a computer to identify a single storage location in its memory.

alphanumeric character — Any digit (zero through nine), or any uppercase or lowercase letter.

American Standard Code for Information Interchange (ASCII) — A standardized code for the characters (letters, digits, punctuation symbols, etc.) stored in a computer. It assigns an integer in the range 0 to 127 to each character. For example, the ASCII code for the letter A is 65. (See Appendix J.)

APL — A high-level programming language developed at IBM in 1962 by K. E. Iverson. It is particularly useful for processing arrays. The name is an acronym for "A Programming Language."

argument — The object of some action. In mathematics, an independent variable is an argument; thus, x is the argument of f(x)=sin x. In programming languages, a parameter is an argument; thus, A is an argument in the procedure call Sort(A,N). In a command language, the objects of a command are called parameters or arguments; thus, TEST1.TXT is an argument of COPY TEST1.TXT TEST2.TXT.

ASCII — [ask'-ee] An acronym for "American Standard Code for Information Interchange."

ASCII file — A file that can be read by humans. Program source code and data files created by the editor are examples of ASCII files. Everything in an ASCII file can be translated meaningfully into characters by means of the ASCII code.

assembler — A program that translates assembly language into machine code.

assembly language — A low-level programming language that is unique to the type of computer. Each assembly language instruction translates into a single machine code instruction.

auxiliary memory — Computer memory residing in devices that are attached to the computer. Magnetic disks and tapes are the most common forms of auxiliary memory. The magnetic disks on VAX computers typically hold hundreds of megabytes. Synonyms: auxiliary storage, mass storage, secondary storage.

BASIC — A high-level programming language developed by John G. Kemeny and T. E. Kurtz at Dartmouth College in 1964. It was designed to be easy for college students to learn. Because most microcomputers provide this language for no extra charge, it has become the most popular programming language. Its name is an acronym for "Beginner's All-purpose Symbolic Instructional Code."

BASIC Mode — A special editing mode on the VAX that allows the user to write and execute BASIC programs the same way as on a microcomputer. This mode is entered from the DCL mode by typing BASIC, and it is exited by typing <CTRL/Z> or EXIT.

batch — Programs that are run without any interactive communication with the user.

baud — A unit for measuring data transmission speed. One baud equals one bit/second. Thus, a 1200-baud modem sends and receives about 150 characters (bytes) per second, which is about six pages per minute. The term honors Emile Baudot (1845–1903) who invented the Baudot code, a predecessor of the ASCII code.

bell — The "beep" sound made by the terminal when it is turned on. The bell may also be programmed to beep at other times. It is actually a nonprintable character whose ASCII code is seven.

bit — The smallest unit of memory in a computer, containing one of two possible values, usually represented by zero and one. If the bit contains a one, it is said that the bit is "set," otherwise, it is "clear." The word is an acronym for "binary digit."

block — The smallest unit of data on secondary storage (disk or tape) that can be transferred to or from the main memory in a computer. On the VAX, one block equals 512 contiguous bytes.

buffer — A temporary storage location, for example to hold lines of text that are to be moved or modified.

byte — [bite] A standard unit of memory in a computer. One byte typically contains eight contiguous bits of information. Because any one of eight bits can be either set or clear (one or zero), each byte may contain any one of 256 (two to the eighth power) values: any number from 0 to 255. This allows the storage of one ASCII character in each byte. Thus, a memory that contains 65,636 bytes (64 K) may store 65,536 characters. Byte is abbreviated "B."

C — A high-level programming language popular among UNIX users. It is like the Pascal language, but it also includes statements for low-level machine control. Because of its efficiency, it is often used for writing operating system programs.

cache memory — A small, intermediate memory location that holds data being transferred between the computer's main memory and the registers of its central processing unit (CPU). By retaining some of the data processed by the CPU, it is able to decrease the processor's access time.

cathode ray tube — The television picture tube that produces the display on the screen of a computer terminal. Also called a video display.

central processing unit — That part of the computer that does the actual computing and controls all of the other parts. The CPUs of most modern computers reside on tiny silicon chips, called "microprocessors."

character — A single symbol: any uppercase or lowercase letter, any of the ten digits (0 through 9), or any of the other symbols found on a typewriter, like a comma (,), an apostrophe ('), an asterisk (*), or even a blank. Each key on a standard keyboard generates a character. The ASCII code is used to store characters in a computer.

character editor — An editor in which each character is the object of the editor's commands.

character string — A contiguous sequence of characters, treated as a single object. For example, 'Good day!' and 'SYS$INPUT' are character strings.

COBOL — [coe'-ball] A high-level programming language which was developed in the late 1950s for business applications. The name is an acronym for "COmmon Business-Oriented Language." Many Cobol instructions are designed specifically for accounting or payroll applications.

code — (1) Rules for interpreting bits, for example, the ASCII code. (2) Line of statements in a program, for example, machine code.

command — A word (e.g. LOGOUT) or phrase (e.g. SHOW TIME) that the computer's operating system or one of its subsystems can understand as an instruction to be executed.

command interpreter — The part of the operating system that translates and executes DCL commands.

command procedure — A file that contains DCL commands. When a command procedure is invoked, its commands execute automatically in sequence.

compile-time error — An error in the source code of a program that is detected by the compiler.

compiler — A program that translates a program written in a high-level language into a program written in machine language. Compilers also provide some diagnostic messages for finding syntax errors. Some compilers even "optimize" the code by reorganizing its parts to make it run faster.

computer — A hardware system that includes a central processor, main memory, and input and output devices.

console terminal — The unique terminal from which the computer operator can start and stop the computer's operating system.

CP/M — An acronym for "Control Program for Microcomputers," a popular operating system produced by Digital Research, Inc. for 8-bit microcomputers.

CPU — An acronym for "Central Processing Unit." This is the brain of a computer, which executes instructions, controls peripherals, and accesses the main memory.

crash — The abrupt self-termination of the execution of a program.

CRT — An acronym for "Cathode Ray Tube."

current direction — The direction in which the cursor is moving while using an editor; forward (toward the end of the file), or backward (toward the beginning of the file).

current line — The reference line on the current page in an editor; usually the line the cursor is on.

current page — The reference page in an editor; usually the page the cursor is on.

current position — The reference position on the current line in an editor; usually the location of the cursor.

cursor — The blinking rectangular light seen on the display screen. (On some terminals, the cursor appears as an underline.) It indicates where the next character will appear.

DCL — An acronym for "Digital Command Language."

DCL Mode — The primary command environment in VAX/VMS. (See Appendix J.)

DCL prompt — The dollar sign ($) that appears on the screen to tell the user that the Digital Control Language is being used, and that the system is ready for a DCL command.

debug — To find and correct errors in a program. (The errors are often called "bugs.")

DEC — [deck] An acronym for "Digital Equipment Corporation."

default — The system's selection of a value of a variable when one is not chosen by the user. For example, when specifying a file to be typed, if the user gives no version number, then the default value is the latest version.

default directory — The current directory; the reference point from which DCL commands are made.

default protection code — The protection code automatically assigned to a file when it is created.

delete access — The ability of a user to delete a file.

delimiter — A symbol used to mark the beginning or end of a string of other symbols. For example, the dot (.) in the filespec TEST1.TXT is a delimiter.

device — A specific piece of computer equipment (hardware). This word is also used to identify that part of a file's specification that precedes the directory part. For example, in the filespec VAX750::DBA0:[REAGANRON]NOTE.TXT;6 the device is DBA0, which refers to a certain disk pack attached to the VAX-11/750 computer where the file is kept.

Digital Command Language — The set of all commands that the VMS operating system can understand and directly execute.

Digital Equipment Corporation — The manufacturer of VAX computers.

directory — A file that contains a list of all the other files in the user's account. It includes the filespecs and other identifications, such as the date on which the file was created.

disk — A circular platter, similar to a phonograph record, that is used for auxiliary storage. It holds data by means of tiny magnetic fields on its oxide surface.

drive — The device on which the computer's external memory is mounted. DEC drives contain either disk cartridges, disk packs, or magnetic tape reels.

echo — To print or display characters that are typed at a computer terminal. For example, terminals normally do not echo passwords.

editor — A large software system, resident on the computer and accessible by all users, which allows the user to create and modify a file; for example, the EDT editor.

editor buffer — The temporary memory location, also called the user's workspace, used to write and modify a file using an editor. When the editor is exited, this space is lost. If the EDT editor is exited using the EXIT command, then the contents of the editor buffer are copied into a new version of the file.

EDT Character Mode — The editing environment internal to the EDT Line Mode which is invoked by the CHANGE command. Also known as the editor's "CHANGE Mode" or "Full Screen Mode."

EDT Line Mode — The editing environment existing when the EDT Editor is first invoked with the EDIT command.

EDT prompt — The asterisk that appears on the screen to tell the user that the EDT Line Editor is in use, and that it is ready for an EDT command.

empty string — The character string with zero characters. It is a synonym of the null string.

end of buffer marker — The symbol [EOB] seen at the end of the file when using the EDT editor. This marker follows the last line of the file.

end of line marker — The unseen marker at the end of each line of a file. This marker is created by the editor when the <RET> key is pressed.

executable image — A machine-language file produced by the VAX/VMS linker. If it is produced from an object module, it will have the same filename as the object module, but its filetype will be EXE.

execute access — The ability of a user to execute a file, e.g., to RUN an executable image or to invoke a command procedure.

field — A variable that makes up one part of a record in a file or database, for example, the zip code field in a file of personnel records.

file — A unit of information kept in a computer's external memory (i.e., on disk or tape). A file is identified by its file specification. Most files contain either programs, parts of programs, data, or text. A listing of all your files is displayed when you give the DCL command DIRECTORY.

file specification — The name that uniquely identifies a file on the VAX. A complete file specification has six parts: its node, device, directory, filename, filetype, and version number. In some organizations (such as the University of Richmond), the logical name SYS$USER is used in place of the device component.

filename — The name the user gives to a file when it is created or renamed. It can contain from 0 to 39 alphanumeric characters, including the hyphen (-) and the dollar sign ($). For example, TEST1 identifies the file as being used to test something.

filespec — An abbreviation for "file specification."

filetype — The extension of the filename that is used to describe how the file is used. For example, the type PAS identifies the file as being part of a Pascal program. It can contain from 0 to 39 alphanumeric characters, including the hyphen (-) and the dollar sign ($).

floating-point operation — An arithmetic operation, such as addition, between two real numbers.

flop — One floating-point operation per second. The word is an acronym for "FLOating-Point operation."

folder — A subfile of the MAIL.MAI file used to store messages classified by the owner.

form feed — The process of a printer advancing to the next page. The form feed character, denoted by <FF>, has ASCII code 12.

Fortran — The first high-level programming language. The name is an acronym for "FORmula TRANslator." It was designed for scientific programming by a team of computer scientists headed by John Backus at IBM. The language has been revised many times since it was first used in 1957. VAX-11 Fortran is an extension of Fortran-77, which is the most common dialect in use. These days, Fortran is used mostly by scientists and engineers.

GB — The abbreviation for "gigabyte."

gigabyte — [guy'-gah-bite] Computer memory storage equal to 1,073,741,824 (or 2^{30}) bytes. Some VAX disk packs hold about half a gigabyte. Some of the new 12-inch optical disks can hold more than one gigabyte.

gigaflop — One billion floating-point operations per second.

group category — The protection code user category that contains all users in the owner's group. The group to which a user belongs is established when the user's account is created. It is identified in your User Identification Code (UIC).

hacker — A person intensely interested in computers and programming, who particularly enjoys discovering hidden aspects of a software system.

hardcopy — The printed copy of a file or part of a file that otherwise would be viewed on the video display terminal screen.

hardware — Computer devices and memory units. The physical components of a computer, such as disks and microprocessors.

HELP file — An on-line source of information about the system within which it resides.

high-level languages — Computer programming languages, like Pascal and BASIC, each of whose statements is translated into many machine-language steps.

I/O — An abbreviation for "Input/Output." It refers to any computer activity or equipment designed to allow information to be transferred to or from the computer. For example, a printer is an I/O device, and READ is an I/O procedure.

Indexed Sequential Access Method — The method for accessing an indexed file that allows the reading or writing of records sequentially.

Insert Mode — The mode that allows line editing of files. Invoked by the EDT command INSERT, essentially all that can be done in this mode is the insertion of text. Return to EDT mode with <CTRL/Z>.

interactive — The status of a program that is interacting with the user as it is running. Programs that run without any concurrent input from, or output to, the user are, instead, said to be in batch mode.

interrupt — An operation, caused either by a user or by the operating system itself, that stops (either temporarily or permanently) the execution of some other process. For example, a user might interrupt the execution of a program by typing <CTRL/Y>, while the operating system will interrupt the execution of a program if an attempt is made to divide by zero.

invoke — To execute a command procedure, by typing the commercial "at" sign (@) followed by the filename of the command procedure.

ISAM — An abbreviation for "Indexed Sequential Access Method."

journal file — A file that contains all the commands entered during an editing session.

K — An abbreviation for "kilobyte."

KB — An abbreviation for "kilobyte."

keypad — The cluster of 18 keys at the far right of the keyboard. These keys are used in the EDT Character Editing mode.

Keypad Mode — Another name for the EDT Character Editing mode, the mode that allows full-screen editing of files. The commands in this mode are

entered using the keypad (the cluster of 18 keys at the far right of the keyboard). (See Appendix C.)

kilobyte — [kill'-o-bite] Computer memory storage equal to 1024 (or 2^{10}) bytes. One kilobyte is the amount of memory needed to store 1024 characters (letters, digits, and other symbols). Microcomputers are often classified according to how many kilobytes of main memory they have. For example, the Commodore 64 computer has 64 kilobytes.

library file — A direct-access file that contains several modules of the same type.

line buffer — A storage area that holds the last line deleted by an editor.

line editor — An editor in which each line of the file is the object of the editor's commands.

line number — A reference number assigned to a line in a file by an editor.

linker — A program invoked by the DCL command LINK, that links a compiled program to other compiled modules, thereby producing an executable image which can then be run.

LISP — A programming language developed by J. McCarthy in 1959. It stands for "LISt Processing." Its most successful applications have been in the area of artificial intelligence research.

log in — To establish contact with the computer's operating system so that it can identify you as a valid user and execute your commands.

log out — To disconnect from the computer's operating system.

logical error — An error in a program that cannot be detected by the computer.

logical name — Another name for one or more parts of a file specification. The user can establish a logical name by means of the ASSIGN command or the DEFINE command.

login file — The command procedure LOGIN.COM that is executed automatically as soon as the user logs in.

machine code — Also called "machine language," it is the language the computer understands, in which all instructions are reduced to strings of bits.

MACRO — (1) The name of the VAX/VMS assembler. (2) The DCL command that invokes the VAX/VMS assembler.

macro — A file that contains commands which execute automatically when the filename of the macro is entered as a command. For example, a command procedure is a macro in the DCL.

MAIL Mode — The mode that allows the user to send and receive "electronic mail" messages from other users. (See Appendix C.)

mainframe computer — A large and expensive (usually more than a million dollars) computer which allows hundreds of terminals. The IBM 4341 and the Burroughs 7000 are common mainframes, usually leased to large corporations, universities, banks, and government institutions like the U.S. Census Bureau.

main memory — The directly addressable physical memory storage which is an internal part of the computer itself. The VAX-11/750 computer at The University of Richmond has a main memory capacity of 6 MB.

MB — An abbreviation for "megabyte."

megabyte — Computer memory storage equal to 1,048,576 (or 2^{20}) bytes. One megabyte typically holds about 660 typewritten pages of text.

megaflop — One million floating-point operations per second.

memory — The physical locations where the computer can store data. Each memory location has an address that identifies it to the computer.

microcomputer — A small and (relatively) inexpensive computer which usually allows only one user at a time. For example, the IBM PC is a microcomputer.

minicomputer — A computer that is larger and faster than microcomputers, but smaller and slower than mainframe computers. DEC is the world's largest manufacturer of minicomputers, and the VAX minicomputers are its most popular.

mip — One million program instructions per second. Continuous, real-time speech recognition requires 200–300 mips.

mode — A particular operating system environment. The operating system maintains a number of system programs, each of which controls the environment in the sense that it determines what commands it will execute. For example, in the EDT Character Editing Mode, the LINE command, executed by pressing the <0> key on the keypad, will cause the cursor to move to the next line of the file. But in DCL mode, this key just types a zero.

Modula-2 — Another high-level programming language invented by Niklaus Wirth, Modula-2 is a direct descendant of Pascal. Its primary advantage over Pascal (and most other languages) is its ability to modularize large software systems into separately compiled modules.

module — A file containing one or more program parts. On the VAX, modules may be compiled independently.

monitor — Another name for the video screen that displays the messages from the computer. Microcomputers are usually attached either to an ordinary television or to a monitor.

NB — An abbreviation for "nanobyte."

nanobyte — [na'-no-bite] Computer storage equal to 1,099,511,627,776 (or 2^{40}) bytes.

network — An interconnected system of computers.

node — An access point to a computer network. Usually, each computer is a separate node. The word also refers to that part of a file specification that identifies the node at which the file is stored.

nonprinting character — A character that cannot be printed as a single character, for example, the form feed character <FF>.

object module — A machine-language file produced from a source code file by a compiler or assembler and used as input to the linker. The module will have the same filename as the source code file, but its filetype will be OBJ.

on-line — The state of being connected to the computer's operating system.

operating system — The master program that operates and controls all aspects of the computer. The main operating system used on VAX computers

is called VMS. UNIX is another popular operating system used on VAXs and other computers. Common operating systems for microcomputers include CP/M, MS-DOS, and PC-DOS (used on the IBM PC).

owner category — The protection-code user category that contains only the owner of the file.

page — A set of 512 contiguous bytes of memory.

page fault — An occurrence in which the operating system has to copy a page from external memory to main memory during the execution of a program.

parameter — Also called an "argument," a parameter is the object of some action. In a command language, the objects of a command are called parameters or arguments; thus TEST1.TXT is a parameter of the command COPY TEST1.TXT TEST2.TXT.

parent directory — The directory of which the current directory is a subdirectory.

Pascal — The high-level programming language named after the French mathematician Blaise Pascal (1623–1662), who is credited with having designed and constructed the first computing device. The language was written by the Swiss computer scientist Niklaus Wirth in 1968. The version used on the VAX is called VAX-11 Pascal, an extension of Standard Pascal.

password — Each user must have a password in order to log in to the VAX. This is to protect the user's account from unauthorized entry. Typically set initially by an organization's Computing Center, passwords should be reset frequently by their owners.

paste buffer — A temporary storage area used in an editor to hold a block of text that is being moved.

PL/1 — A high-level programming language based on Fortran and Algol. The name is an abbreviation for "Programming Language #1."

process — The basic entities recognized by the operating system. A user might have several processes, and a process might exist independent of any user. Processes execute images (programs). Main memory is assigned to processes.

program — A sequence of instructions, usually stored in a computer's memory, which the computer can carry out to manipulate data.

program counter — A register that always contains the address of the next instruction to be executed by the CPU. On the VAX, the program counter is General Register 15 (R15).

prompt — A word, abbreviation, or other symbol displayed by the system to tell the user which mode the system is in and that the system is ready to accept commands for that mode. For example, the prompt for the EDT Line Editing Mode is the asterisk. (See Appendix J.)

protection code — The four-component code assigned to a file when it is created that determines the kinds of access to the file which will be allowed to the various categories of users.

qualifier — A character string (i.e., one or more words connected by underscores) used to modify a DCL command. For example, in the command

DIRECTORY/DATE, the word DATE is a qualifier. In the DCL, qualifiers are preceded by the slash character (/).

quota — The amount of memory allocated to a user.

RAM — An acronym for "Random-Access Memory."

random access — Accessing the components of a table, file, disk, or memory directly, by means of the component's address.

random access memory — That part of a computer's main memory that can be accessed directly (i.e., by means of addresses).

read access — The ability of a user to gain access to a file by means of commands like TYPE and COPY.

read-only memory — That part of the computer's main memory that contains programs that cannot be changed.

real-time — Refers to the interactive execution of a process on a computer, for which the data is "current." The translation of natural-language speech as it is being spoken, or the processing of navigational data on board a space probe in time for it to be used to correct the probe's trajectory, are examples of real-time processes.

register — A storage location adjacent to, and directly accessible by, the CPU. The VAX has sixteen 32-bit registers, named R0 through R15.

reverse video — A display on the screen that is black-on-white instead of the usual white-on-black.

ROM — An acronym for "Read-Only Memory."

root directory — The primary directory in an account. All other directories are subdirectories of it.

run-time error — An error in a program that can cause the program to terminate abnormally (i.e., to "crash").

scrolling — The action of the display lines moving up and off the screen, as if a paper scroll was being unrolled at the bottom and rolled at the top.

search buffer — A temporary storage location used in editors to hold a string of characters being sought.

search string — A string of characters being sought.

select range — A block of text designated while using an editor. With most editors, the select range is determined by marking one end of the range using a SELECT key, and then moving the cursor through the text to the other end of the range.

semantic error — A compile-time error that is not a syntax error.

sequential access — Accessing the components of a list, file, or tape in sequence, one after the other.

sequential file — A file that may only be accessed sequentially.

software — Computer programs and files. Its antonym is hardware.

source code — A text file containing all or part of a computer program written in a programming language like Pascal.

SP — An abbreviation for "Stack Pointer."

stack — An area of main memory that holds references to processes that have been interrupted by other processes.

stack dump — An action taken by the operating system when a program crashes. It reports the contents of the crashing program's stack memory, thereby allowing the user to recover the values of variables at the time of the crash.

stack pointer — A register that always contains the address of the top of the stack. On the VAX, the stack pointer is General Register 14 (R14).

string — A sequence of characters, such as letters and digits, usually marked by delimiters.

subdirectory — A collection of files within an account.

super computer — A very fast (300 – 500 megaflops), very expensive (5 – 15 million dollars) computer. The Cyber 200, built by the Control Data Corporation, is an example. Super computers are found mostly in large research institutions and government installations like the National Security Administration and the U. S. Weather Service.

syntax — The rules that govern a language.

syntax error — A compile-time error caused by a violation of the syntax rules of the programming language.

system — A shortened expression for the computer operating system and its auxiliary software utilities. On the VAX, the operating system is called the Virtual Memory System (VMS).

system category — The protection-code user category that contains only those users authorized by the system manager, usually only the computer operators and system programmers.

system clock — The part of the computer's operating system that keeps the date and time. When the computer is started, the operator enters the current date and time. Thereafter, the system continuously updates them.

system privileges — Privileges accorded those users in the system category. These include the ability to create and access other accounts.

terminal — The peripheral device used to communicate with a timesharing computer system. A terminal usually consists of a keyboard and either a video display screen or a printer.

text buffer — A storage area used by an editor.

text file — An ASCII file; i.e., a file that contains only ASCII characters. A text file may be viewed by means of the TYPE command and may be modified using the EDT editor.

timesharing — Refers to the ability of a computer to serve more than one user at a time. The timesharing activity is masked by the operating system, which switches back and forth very quickly among the users, allocating "time slices" to them.

type-ahead — The ability of the terminal to remember commands typed after the current command, while the current command is still executing. The new commands are not echoed on the screen until they are processed.

UIC — An abbreviation for "User Identification Code."

user — A human being who uses a computer. On the VAX, every user is given a user account, which is identified by its User Identification Code.

User Identification Code — A two-component code assigned to an account when it is created. The first component identifies the user's group, and the second component identifies the member in the group.

username — Each user account has a username, assigned by the system operator when the account is established. The username and its password are required during login.

utility — A program that performs tasks for the system or user. Editors and linkers are utilities.

VAX — The name given by DEC to its series of minicomputers. The word is an acronym for the phrase "Virtual Address eXtension," which refers to the VAX's ability to access over four billion memory cells. (The word "virtual" is used because most of these cells actually reside on disk instead of the computer's main memory.)

version number — Each file has a version number, which appears at the end of its file specification. For example, 6 is the version number of the file specified by TEST1.TXT;6. Each time the EDT editor is used to modify a file, it produces a new version of the file. The PURGE command is used to delete all but the latest version.

virtual address space — A hypothetical array of 4,294,967,296 (or 2^{32}) bytes which an image (program) can address on the VAX.

virtual memory — The actual addressable memory available to the computer's CPU. Although the main memory of the VAX holds only a few megabytes, the computer is capable of addressing over 4 gigabytes (nearly all of which is kept on disks). This is done by rapidly switching 512-byte blocks back and forth between disk and main memory.

VMS — The name of the native operating system used on VAX computers. It is a large software system that operates the computer, managing its timesharing system and allocating its resources (i.e. its memory and its input and output devices). It also keeps track of users' files and runs their programs. VMS is an acronym for "Virtual Memory System."

wildcard — A symbol used to represent other symbols. In the DCL, the asterisk (*) and the percent sign (%) are wildcards. For example, the DCL command TYPE *.PAS would display all files in the user's directory that had the filetype PAS.

workspace — Another name for the user's editor buffer. It is the temporary memory space where the file being edited is kept. With the EDT editor, only one file may be in the user's workspace at any time.

word — On the VAX, two contiguous bytes of memory (16 bits).

world category — The protection-code user category that contains every user on the system.

write access — The ability of a user to modify a file, for example, by means of the EDT editor.

Bibliography

Some books are to be tasted, others to be swallowed, and some few to be chewed and digested; that is, some books are to be read only in parts; others to be read but not curiously; and some few to be read wholly, and with diligence and attention. Some books also may be read by deputy, and extracts made of them by others.

Francis Bacon

This listing separates the DEC manuals from other books. The manuals are available from the Digital Equipment Corporation, Maynard, Massachusetts.

DEC MANUALS

BASIC Pocket Reference Guide, Digital Equipment Corp., No. AA-L341A-TK

BASIC Reference Manual, Digital Equipment Corp., No. AA-L334A-TK

BASIC User's Guide, Digital Equipment Corp., No. AA-L335A-TK

BASIC on VAX/VMS Systems, Digital Equipment Corp., No. AA-L336A-TK

Guide To Text Processing On VAX/VMS, Digital Equipment Corp., No. AA-Y502A-TE

Guide To Using DCL and Command Procedures on VAX/VMS, Digital Equipment Corp., No. AA-Y501A-TE

Introduction to VAX/VMS, Digital Equipment Corp., No. AA-Y500A-TE

VAX EDT Reference Manual, Digital Equipment Corp., No. AA-Z300A-TE

VAX EDT Quick Reference Guide, Digital Equipment Corp., No. AA-Z001A-TE

VAX-11 Pascal Language Reference Manual, Digital Equipment Corp., No. AA-H484C-TE

VAX-11 Pascal Primer, Digital Equipment Corp., No. AA-J180B-TE

VAX-11 Pascal User's Guide, Digital Equipment Corp., No. AA-H485C-TE

Programming in VAX-11 Pascal, Digital Equipment Corp., No. AA-L369A-TE

VAX/VMS Command Definition Utility Reference Manual, Digital Equipment Corp., No. AA-Z408A-TE

VAX/VMS DCL Concepts, Digital Equipment Corp., No. AA-HK73A-TE

VAX/VMS DCL Dictionary, Digital Equipment Corp., No. AA-Z200A-TE

VAX/VMS Mail Utility Reference Manual, Digital Equipment Corp., No. AA-Z421A-TE

VAX/VMS Sort/Merge Utility Reference Manual, Digital Equipment Corp., No. AA-Z431A-TE

OTHER BOOKS

Bell Telephone Laboratories, Inc., *UNIX Time-Sharing System: UNIX Programmer's Manual, Volumes 1-2,* Holt, Rinehart and Winston, 1983 (ISBN 0-03-061742-1).

Bourne, S. R., *The Unix System,* Addison-Wesley, 1982. (ISBN 0-201-13791-7).

Clukey, Lee Paul, *Unix and Xenix Demystified,* TAB Books, 1985 (ISBN 0-8306-0874-5).

Gauthier, Richard, *Using the Unix System,* Reston Publishing, 1981 (ISBN 0-8359-8164-9).

Groff, James R. and Paul N. Weinberg, *Understanding UNIX: A Conceptual Guide,* Que Corp., 1983 (ISBN 0-88022-064-3).

Kernighan, Brian W. and Rob Pike, *The UNIX Programming Environment,* Prentice-Hall, 1984 (ISBN 0-13-937699-2).

Koffman, Elliot B. and Frank L. Friedman, *Problem Solving in Structured BASIC-PLUS and VAX-11 BASIC,* Addison-Wesley, 1984 (ISBN 0-201-10344-3).

Lomuto, Ann Nichols and Nico Lomuto, *A UNIX Primer,* Prentice-Hall, 1983 (ISBN 0-13-937731-X).

Ryan, Barbara F., Brian L. Joiner and Thomas A. Ryan, Jr., *Minitab Handbook*, Second Edition, Duxbury Press, 1985 (ISBN 0-87150-470-7).

Ryan, Jr., Thomas A., Brian L. Joiner and Barbara F. Ryan, *Minitab Reference Manual*, 1982.

Shannon, Terry C., *Introduction To VAX/VMS*, Professional Press, 1985 (ISBN 0-9614729-0-1).

Shpuntoff, Albert F., *The UNO VAX Book*, Kendall/Hunt, 1985 (ISBN 0-8403-3759-0).

Silvester, Peter P., *The Unix System Guidebook, An Introductory Guide for Serious Users*, Springer-Verlag, 1984 (ISBN 0-387-90906-0).

Thomas, Rebecca, and Jean Yates, *A User Guide to the Unix System*, Osborne/McGraw-Hill, 1982 (ISBN 0-931988-71-3).

Townsend, Clifford, *VAX BASIC*, Holt, Rinehart and Winston, 1986 (ISBN 0-03-921848-1).

Index

Index

If you don't find it in the Index, look very carefully through the entire catalog.

Sears, Roebuck and Co. (1897)

A

abbreviated commands, 37
access, types of, 110
ADD command, 76
APPEND command, 30, 31, 32
arguments, 23, 133
arithmetic mean, 73
array processing, 83
ASCII code, 79, 201-204
asterisk, 105

B

BASIC commands, 89, 183-185
BASIC mode, 88, 155
BASIC program, 89
  debugging, 90
  editing, 91
  interrupting, 98
  logical errors and, 96
  running, 90, 92
  stopping, 98
  syntax errors and, 95
  text files in, 94
  using EDT editor with, 94
  VAX mode of, 88
  versions of, 87
  writing and listing of, 90
batch files, 115
blocks, 47
BRIEF qualifier, 43

C

CENTER command, 77
CHANGE command, 42
character editor, 41
  commands for, 44
column labels, 71
COMMAND command, 49
command procedures, 115
  aborting, 118
  comments for, 117
  debugging with, 119
  interactive input with, 124
  ON . . . THEN command for, 120
  parameter of, 120
  producing output with, 123
  repetitive tasks with, 123
  responding to errors in, 120
  user interaction with, 117
  uses for, 116
compile-time error, 57, 62
compiled BASIC, 88
compiler, 57
concatenation, 32, 134
continuation symbol, 124
COPY command, 23, 30, 31, 32, 33, 105
CORRELATE command, 75
correlation coefficient, 75
crash, 66
CREATE command, 15, 102
CREATE/DIRECTORY command, 102
current line, 17
cutting and pasting, 47

D

DCL commands, 7
DCL mode, 6, 18, 19, 42, 88, 153
  commands for, 157-162
  manipulating files with, 29
debugging, 91

declaration section, 61
default directory, 103
default protection code, 110
DEFAULT qualifier, 111
DEFINE/KEY command, 38
DEFINE/KEY/TERMINATE command, 38
DELETE command, 29, 30, 106
delete commands, 46
DESCRIBE command, 74
device, 14
directory, 14
DIRECTORY command, 16
directory file, 102
DIRECTORY/SIZE command, 33, 34

E

ed editor, 131
EDIT command, 97
editing buffer, 21
editor, 16
EDT editor, BASIC program and, 94
EDT keypad commands, 169-174
EDT keypad editor, 41
EDT line editor, 13
  creating and referencing files with, 15
EDT line mode, 18, 19, 42, 88, 153
  commands for, 163-167
ellipsis, 106
emulator, 128
ERASE command, 82
executable image, 58
EXIT command, 89
extension, 14

F

file protection, 108
file specification, 13
filename, 14
files
  combining, 33
  creating and referencing of, 15
  manipulation of, 29
filetype, 14
FILL command, 50, 52
FIND command, 49, 50
FIND NEXT command, 50
fixed-point notation, 65
floating-point notation, 65
full-screen editor, 41

G

grep command, 136
group code, 109

H

HELP command, 10
HELP SET PASSWORD command, 10
histogram, 74

I

identifier, 61
INFORMATION command, 72
INQUIRE command, 125
INSERT command, 17, 20
INSERT mode, 42, 88
interactive input, 124
interactive program, 61
interpreted BASIC, 88
interpreter, 88
interrupt, 9
inverse commands, 80

J

journal file, 25

K

kernel, 187
keypad, defining keys on, 37
keypad map, 44, 45
keypad mode, 41, 42, 43, 88

L

label, 120
line codes, 21
line editing mode, 16
line editor, 140
LINK command, 58
logical errors, 63, 96
logical name, 60
login procedure, 121

M

machine language, 57
MAIL mode, 156
  commands for, 197-200
main block, 56
main buffer, 27
median, 73
member code, 109
memory, management of, 33
Minitab
  ADD command for, 76
  arithmetic mean with, 73
  array processing with, 83
  ASCII and non-ASCII files on, 79
  CENTER command for, 77
  column labels for, 71
  commands for, 175-181
  correlation coefficient with, 75
  creating new columns in, 76
  data for, 71
  default files in, 80
  DESCRIBE command for, 74
  histogram with, 74
  INFORMATION command for, 72
  introduction to, 69
  NAME command for, 72
  PLOT command for, 75
  plotting data with, 75
  PRINT command for, 72
  READ command for, 71
  retrieving data with, 82
  SAVE command for, 78
  simulation with, 84
MINITAB command, 70
Minitab mode, 69, 154
modes, interaction of, 18
MOVE command, 22

N

NAME command, 72
native mode, 128
node, 14

O

object module, 57
ON . . . THEN command, 120
operating system, 127
options, 132

P

page delimiters, 46
parameter, 120
parent directory, 103
PASCAL command, 57
Pascal program
  compiling, 57
  debugging, 64, 65
  interactive feature of, 61
  language of, 55
  linking, 58
  running, 59
  sample of, 56
  variables for, 61
  writing, 56
password, 5, 6, 7
PASTE buffer, 48
PLOT command, 75
PRINT command, 72
program heading, 56
protection codes, 101, 110
PURGE command, 29, 30

Q

qualifiers, 27
quota, 33, 35

R

RANDOM command, 84
random number function, 84
range, 141
RANK command, 82
READ command, 71, 81
readln procedure, 62
RECOVER qualifier, 26
reformatting, 48
repeat loop, 62
RESEQUENCE command, 24
RETRIEVE command, 79
root directory, 103
run-time errors, 66, 91

S
SAVE command, 78, 93, 94
scope, 105
search buffer, 50
search codes, 21
SELECT command, 47, 50, 52
select range, 47
SEQUENCE command, 89
sequence mode, 88
SET DEFAULT command, 102
SET PROTECTION command, 108, 111, 112
  syntax for, 112
SET VERIFY command, 117
severity level of error, 63
shell commands, 187
shell procedures, 115
SHOW command, 116
SHOW DEFAULT command, 102
SHOW QUOTA command, 35
SHOWTIME command, 8
simulation, 84
SORT command, 82
standard deviation, 73
statistical analysis, 70
string assignment statements, 122
string variable, 98
subcommand, 84
subdirectories, 101
SUBSTITUTE command, 24, 51
symbolic stack dump, 66
syntax errors, 57, 95

T
terminal, 1
terminal session, 7
text files, 94
tree diagram, 103, 104
type, 14
TYPE command, 16
TYPE WHOLE command, 17, 43

U
UNIX
  arguments with, 133
  concatenating files with, 134
  ed editor of, 131
  file processing with, 130
  fundamental commands for, 129
  grep command of, 136
  logging in to, 129
  operating system of, 128
  options of, 132
  prompt for, 129
  shell commands for, 187-192
  transition to, 127
  wildcards of, 133
UNIX mode, 155
UNIX text editor, 139
  commands for, 141
  commands for, 141, 193-195
  creating a file with, 141
  deleting and inserting lines with, 144
  inserting external files with, 143
  line editor of, 140
  MOVE command for, 145
  SEARCH command for, 145
  substitute command for, 144
  syntax of, 141
user categories, 109
user errors, 8
user identification code, 109
username, 4

V
value, 108
variable, 61
VAX
  DCL commands for, 7
  DCL mode of, 6
  logging in, 4
  logging out, 7
VAX modes, 151-156
VAX/VMS subdirectories and protection codes, 101
version, 14
VT100 keyboard, 2
VT200 keyboard, 3

W
wildcards, 36, 133
Wirth, Niklaus, 55
word buffer, 46
workspace, 71
WRITE command, 79
write list, 124
writeln procedure, 62
wraparound, 43

PF1 GOLD	PF2 HELP	PF3 FNDNXT FIND	PF4 DEL L UND L
7 PAGE COMMAND	8 SECT FILL	9 APPEND REPLACE	- DEL W UND W
4 ADVANCE BOTTOM	5 BACKUP TOP	6 CUT PASTE	, DEL C UND C
1 WORD CHNGCASE	2 EOL DEL EOL	3 CHAR SPECINS	ENTER ENTER SUBS
0 LINE OPEN LINE		. SELECT RESET	

Send for FREE TAB Catalog describing over 1200 current t

TAB Professional and Reference Books
Division of TAB BOOKS Inc.
P.O. Box 40, Blue Ridge Summit, PA 17214

F
ISBN 0-
PRICES HIGHER I